НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

До захисту допущено:
Завідувача кафедри
_____ О.Л. Тимощук
«___»_____20__ р.

Дипломна робота
на здобуття ступеня бакалавра

за освітньо-професійною програмою «Бакалавр»

спеціальності 124 "Системний аналіз"

на тему: «Задача знаходження інформаційного блоку на веб сторінці на основі машинного навчання»

Виконав:
студент IV курсу, групи КА-61
Якубенко Олексій Петрович _____

Керівник:
Доцент, к.т.н. Дідковська М.В. _____

Консультант з економічного розділу:
Доцент, к.е.н. Шевчук О.А. _____

Консультант з нормоконтролю:
Доцент, к.т.н. Коваленко А. Є. _____

Рецензент:
Доцент, к.т.н. Т. М. Заболотня _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 «Системний аналіз»

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
В.о.завідувача кафедри
_____ Оксана ТИМОЩУК
«___»_____20__ р.

ЗАВДАННЯ

на дипломну роботу студенту

Якубенко Олексій Петрович

1. Тема роботи «Задача знаходження інформаційного блоку на веб сторінці», керівник роботи Дідковська Марина Віталіївна, доцент, к. т. н., затверджені наказом по університету від «_25_»_травня_ 20_20_ р. №_1143-с__

2. Термін подання студентом роботи 08 травня 2020 року _____

3. Вихідні дані до роботи

4. Зміст роботи

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

6. Консультанти розділів роботи*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
4	доцент к. е. н., Шевчук О. А.		29.05.2020

7. Дата видачі завдання _____

* Якщо визначені консультанти. Консультантом не може бути зазначено керівника дипломної роботи.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримане завдання	13.04.2020	Виконано
2	Збір інформації	20.04.2020	Виконано
3	Ознайомлення з літературою	27.04.2020	Виконано
4	Аналіз вимог завдання і вибір засобів розв’язання поставленної задачі	04.05.202	Виконано
5	Розробка програмного продукту	11.05.2020	Виконано
6	Виконання функціонально-вартісного аналізу	29.05.2020	Виконано
7	Отримання висновків після тестування	30.05.2020	Виконано
8	Оформлення дипломної роботи	08.06.2020	Виконано
9	Оформлення допуску до захисту та подача роботи до ДЕК	12.06.2020	Виконано

Студент Якубенко Олексій

Керівник Марина Дідковська

РЕФЕРАТ

Дипломна робота: 89 с., 8 табл., 26 рис., 2 дод, 19 джерел.

ЗНАХОДЖЕННЯ ІНФОРМАЦІЙНОГО БЛОКУ НА ВЕБ СТОРІНЦІ.

Об’єкт дослідження: алгоритм знаходження інформаційного блоку на веб сторінці.

Предмет дослідження: методи знаходження інформаційного блоку на веб сторінці, що базуються на текстовому аналізі та html аналізі

Мета дослідження: створити програмний продукт для створення інструменту знаходження інформаційного блоку на веб сторінці

Постановка задачі: розробити флгоритм знаходження інформаційного блоку на веб сторінці на базі нейронних мереж

Програмний продукт було розроблено з використанням мови програмування Python.

ABSTRACT

Thesis: 89 p., 8 tabl., 26 fig., 2 append., 19 sources.

INFORMATIVE BLOCK EXTRECTION FROM WEB PAGES.

Object of research: algorithm for informative block extraction from web pages

Subject of research: methods of finding an information block on a web page based on text analysis and html analysis

The purpose of the study: create a software product that can create a tool to find the information block on a web page

Problem statement: develop an algorithm for finding an information block on a web page based on neural networks

The software product was developed using Python programming language.

ЗМІСТ

ВСТУП

На сьогоднішній день, в еру розвитку інформаційних інтернет джерел необхідно своєчасно в автоматичному режимі отримувати максимально повну та структуровану інформацію. Алгоритм автоматичного вилучення потрібної інформації з веб сторінок є корисним інструментом для цього завдання.

Об’єктом досліджень є методи текстового аналізу та методи аналізу структури веб сторінки.

Метою роботи є створення python бібліотеки для створення інструменту який буде знаходити довільну необхідну користувачу інформацію на веб сторінці на основі глибоких нейронних мереж.

Актуальність такого інструмента зумовлена потребою інформаційних агрегаторів, компаній які займаються аналізом ризиків та консультуванням в автоматизації збору інформації яка наявна в інтернет мережі як для оцінки поточної ситуації так і для створення баз даних.

РОЗДІЛ 1 ЗАДАЧА ЗНАХОДЖЕННЯ ІНФОРМАЦІЙНОГО БЛОКУ НА ВЕБ СТОРІНЦІ. ІСНУЮЧІ МЕТОДИ ЇЇ ВИРІШЕННЯ. АНАЛІЗ РОБОТИ НАЯВНИХ ДОСТУПНИХ ПЛАТФОРМ

1.1 Задача знаходження інформаційного блоку

Потреба у виникненні технологій, які б дозволили діставати з сайту потрібну інформації з’явилась з утворенням інтернету. І по сьогоднішній день ця тема є актуальною для дослідження, адже вона допомагає вирішувати велику кількість прикладних задач.

В цій роботі першочерговим завданням є розв’язання проблеми знаходження необхідної користувачу інформації на веб сторінці, бо робота з веб сторінкою має особливість неструктурованості в тестовому плані через те, що вона обробляється в html форматі[1].

Наприклад, можливо в режимі онлайн збирати текс новин з багатьох сайтів новин або збирати опис товарів з інтернет магазинів.

Завдання знаходження потрібної інформації на веб сторінці можна розглядати як проблему класифікації html блоків за допомогою текстового аналізу та аналізу знаходження html блоку в html структурі, де класами є потрібна інформація та непотрібна інформація.

На сьогоднішній день задача знаходження потрібної інформації на веб сторінці вже має варіанти розв’язку. Існують різні способи отримати потрібну інформацію з веб сторінки, проте всі вони мають певні недоліки. Також існує велика кількість бібліотек для роботи з структурою html та нейронних мереж створених для вирішення цієї проблеми.

Проте кожен з цих підходів має свої недоліки. Тому попри існування вже розроблених методів, створення нових методів знаходження необхідної

користувачу інформації на веб сторінці актуально і зараз. Тому слід розібрати всі існуючі підходи до цієї задачі, та розробити свій покращений метод.

Існує два основних способи вирішення задачі знаходження необхідної інформації на веб сторінці:

- Фіксований шлях до інформації в html структурі сайту
  - Шлях треба знаходити для кожного сайту окремо в ручну
  - При зміні html структури сайту треба оновлювати шлях в ручну
- Аналіз тексту веб сторінки
  - Не використовує html структуру сайту
  - Потребує великих обчислювальних витрат

1.2 Існуючі методи вирішення задачі знаходження потрібної інформації на веб сторінці

1.2.1 Фіксовані шляхи в html структурі сайту

Одним з класичних методів знаходження потрібної інформації на веб сторінці є фіксовані шляхи в html структурі веб сторінки. Основою цього методу є сам формат html. Html представляє з себе дерево тегів де листи цього дерева несуть в собі текстову і не тільки інформацію яку бачить користувач, тобто якщо ви знаєте шляху у цьому дереві до потрібної вам інформації, ви легко зможете її отримати. Наприклад для сайту новин можна встановити шлях до тексту новини, так як зазвичай всі веб сторінки з новинами мають однакову структуру в межах одного сайту ви зможете отримати текст усіх новин сайту. Проте з часом html структура сайту змінюється в силу зміни дизайну, додання рекламних банерів і ще багатьох причин, тому потрібно буде вручну оновлювати цей шлях при кожній такій зміні.

Звичайно кожен сайт зазвичай має свою унікальну структуру, а тому для кожного окремого сайту доведеться знаходити html в ручну.

1.2.2 Аналіз тексту

Зазвичай виконується за допомогою класичного машинного навчання або нейронних мереж. Основна відмінність методів машинного навчання полягає у тому, що нейронні мережі та моделі класичного машинного навчання не програмуються, а навчаються.

Для навчання таким методам потрібно заздалегідь підготовленні данні у вигляді пар інформація з якої потрібно зробити висновок та правильний висновок. Тобто в нашій задачі це будуть пари веб сторінка та текст, де текст – це та необхідна інформація яку ми хочемо знаходити на веб сторінці. Веб сторінка подається у вигляді набору чи одного тесту, який в свою чергу токінезується.

Такі методи потребують дуже великої кількості даних для точної роботи, що іноді ставить під сумнів доцільність їх використання.

Один з недоліків таких методів полягає у тому що неможливо отримати сто відсотків правильні відповіді. Проте ці методи здатні вирішувати задачі які класичні методи вирішити зовсім не можуть або можуть з над великими обчислювальними витратами та потребою дуже великого часу на розробку.

Одним з плюсів таких методів є легкість адаптування до нових умов (наприклад з плином часу). Достатньо просто зібрати нові данні та перенавчити вже існуючу модель.

1.3 Аналіз та порівняння платформ для знаходження інформаційного блоку

1.3.1 Фіксовані шляхи в html структурі сайту

XPath (XML Path Language) - мова запитів до елементів XML-документа[2]. Розроблено для організації доступу до частин документа XML в файлах трансформації XSLT[3] і є стандартом консорціуму W3C. XPath покликаний реалізувати навігацію по DOM в XML. У XPath використовується компактний синтаксис, відмінний від прийнятого в XML. У 2007 році завершилася розробка версії 2.0, яка тепер є складовою частиною мови XQuery 1.0. У грудні 2009 року почалася розробка версії 2.1, яка використовує XQuery 1.1.

Мова XPath заснована на представленні дерева документа XML та забезпечує можливість переміщення по дереву, вибираючи вузли за різними критеріями.

Найважливіший вид виразу в XPath - це місце розташування. Шлях розташування складається з послідовності кроків розташування. Кожен етап розташування містить три компоненти:

- вісь
- визначення вузла
- предикати (може бути декілька, або не бути взагалі).

Вираз XPath оцінюється відносно контекстного вузла. Специфікатор осі, такий як "дочірній" або "нащадок", визначає напрямок для переміщення з контекстного вузла. Тест вузла та предикат використовуються для фільтрації вузлів, визначених специфікатором осі: Наприклад, для тестування вузла "A" потрібно, щоб усі вузли, на яких переходили, мали мати позначку "A". Предикат можна

використовувати, щоб вказати, що вибрані вузли мають певні властивості, які задаються самими виразами XPath.

На початку покликаний надати загальний синтаксис і модель поведінки між XPointer та XSLT, XPath швидко здобув визнання розробників як мова малих запитів

Приклад використання:

Простий XML документ (рис. 1.1):

```xml
<?xml version="1.0" encoding="utf-8"?>
<Wikimedia>
  <projects>
    <project name="Wikipedia" launch="2001-01-05">
      <editions>
        <edition language="English">en.wikipedia.org</edition>
        <edition language="German">de.wikipedia.org</edition>
        <edition language="French">fr.wikipedia.org</edition>
        <edition language="Polish">pl.wikipedia.org</edition>
        <edition language="Spanish">es.wikipedia.org</edition>
      </editions>
    </project>
    <project name="Wiktionary" launch="2002-12-12">
      <editions>
        <edition language="English">en.wiktionary.org</edition>
        <edition language="French">fr.wiktionary.org</edition>
        <edition language="Vietnamese">vi.wiktionary.org</edition>
        <edition language="Turkish">tr.wiktionary.org</edition>
        <edition language="Spanish">es.wiktionary.org</edition>
      </editions>
    </project>
  </projects>
</Wikimedia>
```

Рисунок 1.1 – XML документ

Приклади XPath виразів:

```
/Wikimedia/projects/project/@name
```

вибирає атрибут name для всіх <project>

```
/Wikimedia//editions
```

вибирає всі <edition> всіх <project>

```
/Wikimedia/projects/project/editions/edition[@language='English']/text()
```

вибирає адреси всіх англійських проектів Wikimedia (текст усіх елементів, де атрибут мови дорівнює англійській мові).

```
/Wikimedia/projects/project[@name='Wikipedia']/editions/edition/text()
```

вибирає адреси всіх Wikipedias (текст усіх елементів видання, які існують під елементом project, з атрибутом імені Wikipedia).

1.3.2 Аналіз тексту

1.3.2.1 TFIDF

У пошуку інформації, tf – idf або TFIDF, що скорочується на термін «term frequency–inverse document frequency»[4], є числовою статистикою, яка призначена для відображення того, наскільки важливим є слово для документа в колекції або корпусі. Він часто використовується як коефіцієнт зважування для пошуку інформації, пошуку тексту та моделювання користувача. Значення tf – idf збільшується пропорційно кількості разів, коли слово з’являється в документі, і компенсується кількістю документів у корпусі, що містять слово, що допомагає налаштувати на те, що деякі слова взагалі з’являються частіше. tf – idf - одна з найпопулярніших схем зважування токенів на сьогодні. Опитування, проведене в 2015 році, показало, що 83% текстових систем рекомендацій у цифрових бібліотеках використовують tf – idf.

Варіанти схеми зважування tf – idf часто використовуються пошуковими системами як центральний інструмент для оцінювання та ранжування відповідності документа з огляду на запит користувача. tf – idf може бути успішно використаний

для фільтрації слів у різних тематичних полях, включаючи узагальнення тексту та класифікацію.

Одну з найпростіших функцій ранжування обчислюють шляхом підсумовування tf – idf для кожного токена запиту. Багато більш складних функцій ранжирування є варіантами цієї простої моделі.

Для задачі класифікації TF-IDF використовують як векторну модель, результат роботи якої (вектори) вже класифікують інші алгоритми (наприклад логістична регресія).

Порівняно з нейронними мережами пара TF-IDF + логістична регресія[5] потребує набагато менше обчислювальних ресурсів та швидше навчається. Проте розглядає текст як набір слів, а не як послідовність через що зазвичай результат роботи гірше ніж у нейронних мереж.

1.3.2.2 Рекурентні нейронні мережі

Рекурентна нейронна мережа (RNN) - це клас штучних нейронних мереж, де з'єднання між вузлами утворюють спрямований графік по часовій послідовності[6]. Це дозволяє йому проявляти часову динамічну поведінку. На відміну від класичних нейронних мереж, рекурентні нейронні мережі можуть використовувати свій внутрішній стан (пам'ять) для обробки послідовностей змінної довжини.

Термін "рекурентна нейронна мережа" використовується без розбору для позначення двох широких класів мереж з подібною загальною структурою, де одна є кінцевим імпульсом, а інша - нескінченним імпульсом. Обидва класи мереж демонструють часову динамічну поведінку. Кінцева імпульсна рекуррентна мережа - це спрямований ациклічний графік, який можна розкрутити і замінити класичною нейронною мережею, тоді як нескінченна імпульсна рекуррентна мережа - це спрямований циклічний графік, який неможливо розкрутити.

Як кінцеві імпульсні, так і нескінченні повторювані імпульсні мережі можуть мати додаткові збережені стани, і це сховище може знаходитися під прямим контролем нейронної мережі. Сховище також може бути замінено іншою мережею або графіком, якщо це включає затримки в часі або має петлі зворотного зв'язку. Такі керовані стани називаються затворними або закритими пам'яттю і є частиною Дóвго-короткочáсних мереж пам'яті (LSTM[7]).

1.3.2.3 Трансформери

Трансформатор - це модель глибокого машинного навчання, запроваджена в 2017 році, яка використовується насамперед у галузі обробки природних мов (NLP)[8]. Як і рекурентні нейронні мережі (RNN), трансформатори призначені для обробки упорядкованих послідовностей даних, таких як природний мова, для різних завдань, таких як машинний переклад та узагальнення тексту. Однак, на відміну від рекурентних нейронних мереж, трансформатори не вимагають, щоб послідовність оброблялася по порядку. Отже, якщо дані, про які йдеться, є природною мовою, Трансформатору не потрібно обробляти початок речення до того, як він обробляє кінець. Завдяки цій особливості Трансформатор дозволяє набагато більше паралелізації, ніж рекурентні нейронні мережі під час тренування.

З моменту їх введення Трансформери стали основним складовим елементом більшості сучасних архітектур в обробці природної мови, замінивши в багатьох випадках рекурентні нейронні мережі, такі як довготривала-короткочасова пам'ять (LSTM). Оскільки архітектура Трансформатора сприяє більшій паралелізації під час навчальних обчислень, вона дозволила навчатись на набагато більше даних, ніж це було можливо до її введення. Це призвело до розробки попередньо натренованих систем, таких як BERT[9] та GPT-2[10], які пройшли навчання з величезною

кількістю загальних мовних даних, а потім можуть бути тонко налаштовані та підготовлені до конкретного завдання.

1.4 Обґрунтування вибору методу

Оскільки аналіз тексту видає найкращі результати в цій задачі, але при цьому не використовує html структуру веб сторінки (яку також можна трансформувати в послідовність для кожного окремого блоку) будо прийнято рішення поєднати аналіз тексту та аналіз структури html веб сторінки в один продукт який буде реалізовано завдяки нейронним мережам.

Для цього доведеться розробити алгоритм/архітектуру нейронної мережі обробки html структури сторінки.

1.5 Висновки за розділом

Було проведено аналіз існуючих готових рішень для вирішення поставленої задачі знаходження потрібної інформації на веб сторінці, та зроблено висновок щодо актуальності створення спеціалізованого модулю.

Також було проаналізовано існуючі способи та алгоритми, які використовуються для вирішення задачі знаходження інформаційного блоку з веб сторінки, та визначено найкращий метод для виконання даної роботи.

РОЗДІЛ 2 МАТЕМАТИЧНІ ОСНОВИ НЕЙРОННИХ МЕРЕЖ ДЛЯ АНАЛІЗУ ТЕКСТУ

2.1 Методи

2.1.1 Токенізація

Токенізація - це процес поділу кількості тексту на менші частини, що називаються токенами[4].

Іншими словами , токенізація - це процес розбиття заданого тексту на одиниці, що називаються токенами. Токенами можуть бути слова чи цифри або розділові знаки. Токенізація виконує це завдання, заходячи межі слів. Кінцева точка слова та початок наступного слова називають межами слова.

Враховуючи послідовність символів та визначений документ, токенізація - це завдання розрізати його на шматки, що називаються токенами, можливо, одночасно викидаючи певні символи, наприклад, пунктуацію. Ось приклад токенізації (рис. 2.1):

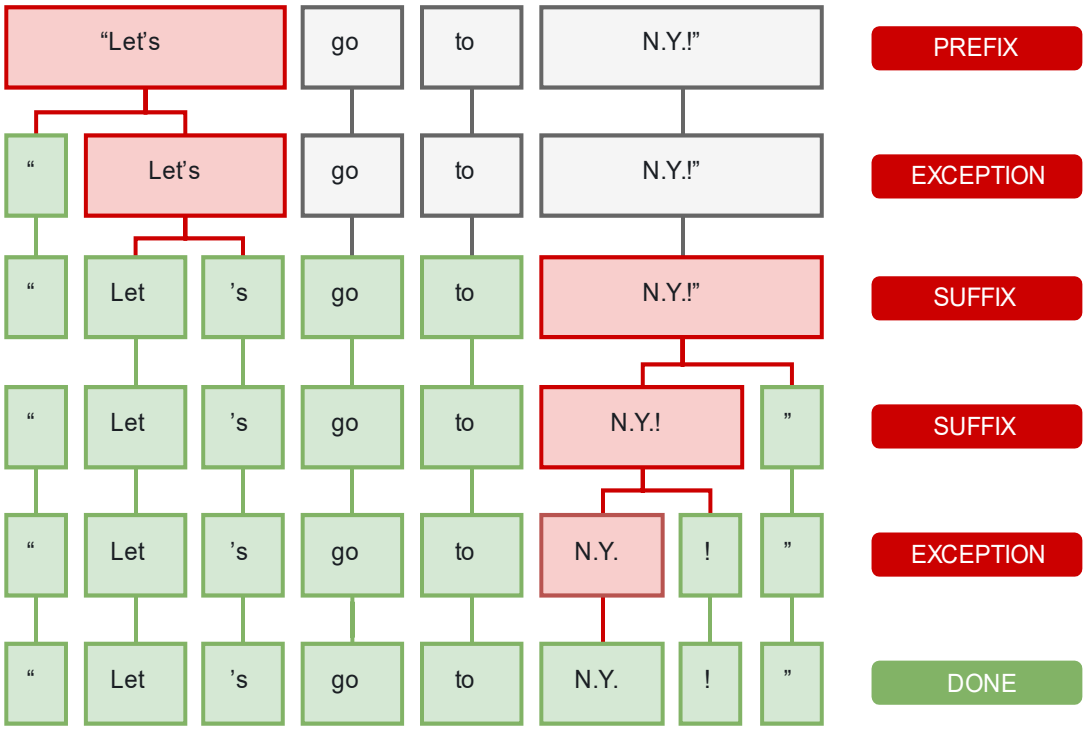


Рисунок 2.1 – приклад токенізації

На сьогоднішній час існує багато готових програмних рішень для токінезації. Приклад токінезації з python бібліотеки nltk (рис. 2.2):

```
from nltk.tokenize import word_tokenize

EXAMPLE_TEXT = "Hello Mr. Nitin, what are you doing today?"
print(word_tokenize(EXAMPLE_TEXT))
print(len(word_tokenize(EXAMPLE_TEXT)))
```

```
['Hello', 'Mr.', 'Nitin', ',', 'what', 'are', 'you', 'doing', 'today', '?']
10
```

Рисунок 2.2 – Приклад токінезації з python бібліотеки nltk

2.1.2 Ембеддінги

Ембеддінги - це відображення дискретної - категоріальної - змінної до вектора раціональних чисел[11]. У контексті нейронних мереж ембедінги є маломірними раціональними векторними відображаючими дискретні змінні. Ембедінги в нейронних мережах корисні, оскільки можуть зменшити розмірність категоричних змінних і інформативно представляти категорії в трансформованому просторі.

Ембедінги в нейронних мережах мають три основні цілі:

- Пошук найближчих сусідів у вбудованому просторі. Вони можуть бути використані для створення рекомендацій на основі інтересів користувачів або категорій кластерів.
- Як вхід до моделі машинного навчання для завдач навчання з вчителем.
- Для візуалізації відносин та залежностей між категоріями.

Розглянемо ембедінги в контексті реальної проблеми: представити всі книги у Вікіпедії як вектори для створення системи рекомендацій щодо книг (рис. 2.3).

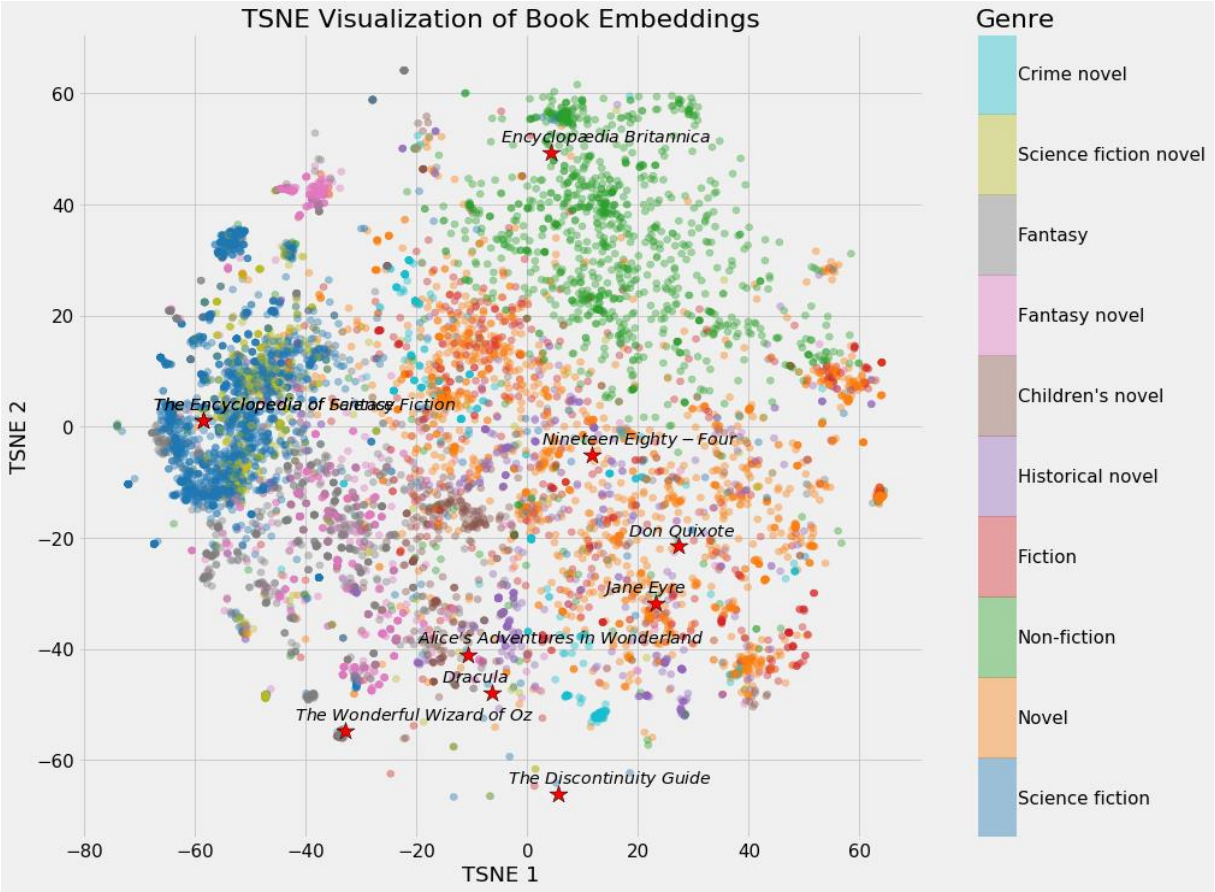


Рисунок 2.3 – системи рекомендацій книг

2.1.3 GRU мережа

Розглядаючи підходи машинного навчання для аналізу тексту слід розглянути рекурентні нейронні мережі[12].

Gated recurrent units (GRU), є механізмом воріт в рекурентних нейронних мережах, запроваджених у 2014 році Кюнґюном Чо та ін. GRU - це як довга короткочасна пам'ять (LSTM) з воротами забуття, але має менші параметри, ніж LSTM, оскільки їй не вистачає вихідних воріт. Результати GRU щодо певних завдань моделювання поліфонічної музики, моделювання мовного сигналу та обробки природних мов виявилося подібним до показників LSTM. Показано, що GRU демонструють ще більшу ефективність на деяких менших наборах даних ніж LSTM.

Однак, як показали Гейл Вайс, Йоав Голдберг та Еран Яхав, LSTM є "строго сильнішим", ніж GRU, оскільки він може легко проводити безлімітний підрахунок, тоді як GRU не може.

Існує декілька варіацій архітектури для клітини GRU, в яких ворота використовують попередній прихований стан та зміщення в різних комбінаціях, а також спрощена форма, яка називається мінімальною одиницею.

Fully gated unit (рис 2.4)

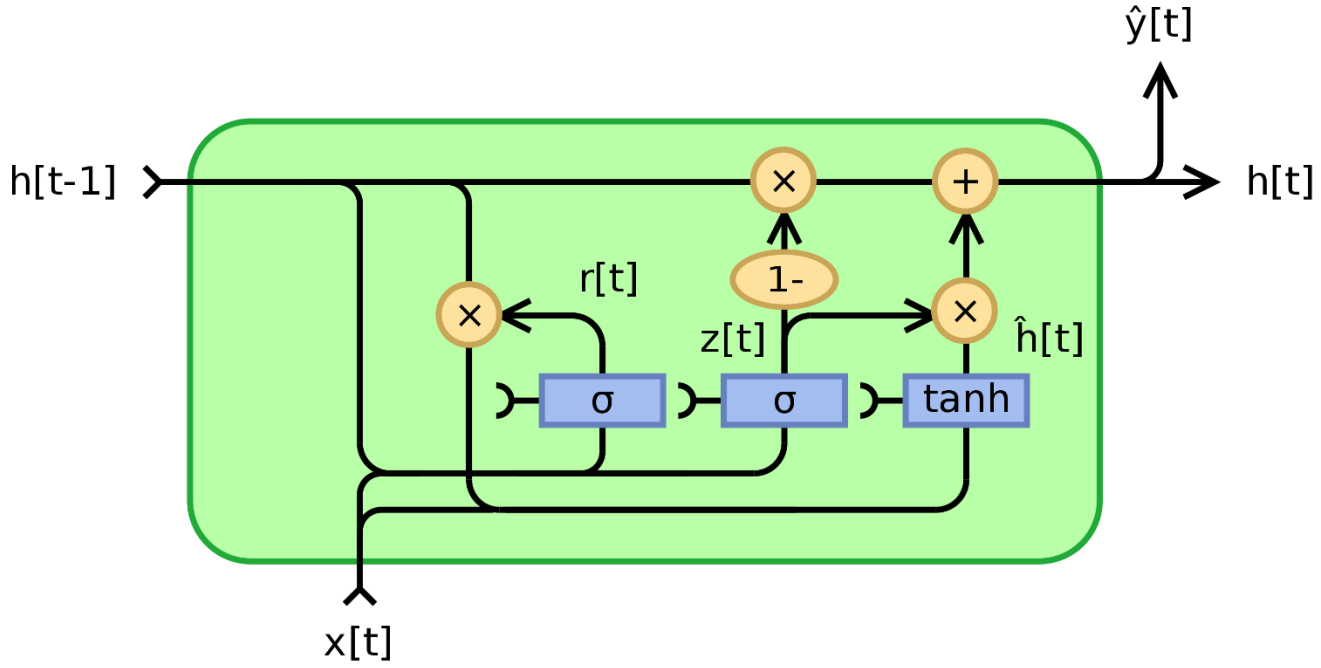


Рисунок 2.4 – схема Fully gated unit

Для t = 0 вихідний вектор h = 0

$$
\begin{aligned}
z_t &= \sigma_g(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma_g(W_r x_t + U_r h_{t-1} + b_r) \\
h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \phi_h(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)
\end{aligned}
\tag{2.1}
$$

Де

- x_t: input vector
- h_t: output vector
- z_t: update gate vector
- r_t: reset gate vector
- W, U and b: parameter matrices and vector

Функції активації

- σ_g: The original is a sigmoid function.
- ϕ_h: The original is a hyperbolic tangent.

Альтернативні архітектури можуть бути створені змінюючи z та r

– Тип 1, кожені ворота залежать лише від попереднього прихованого стану та зміщення(рис 2.5)

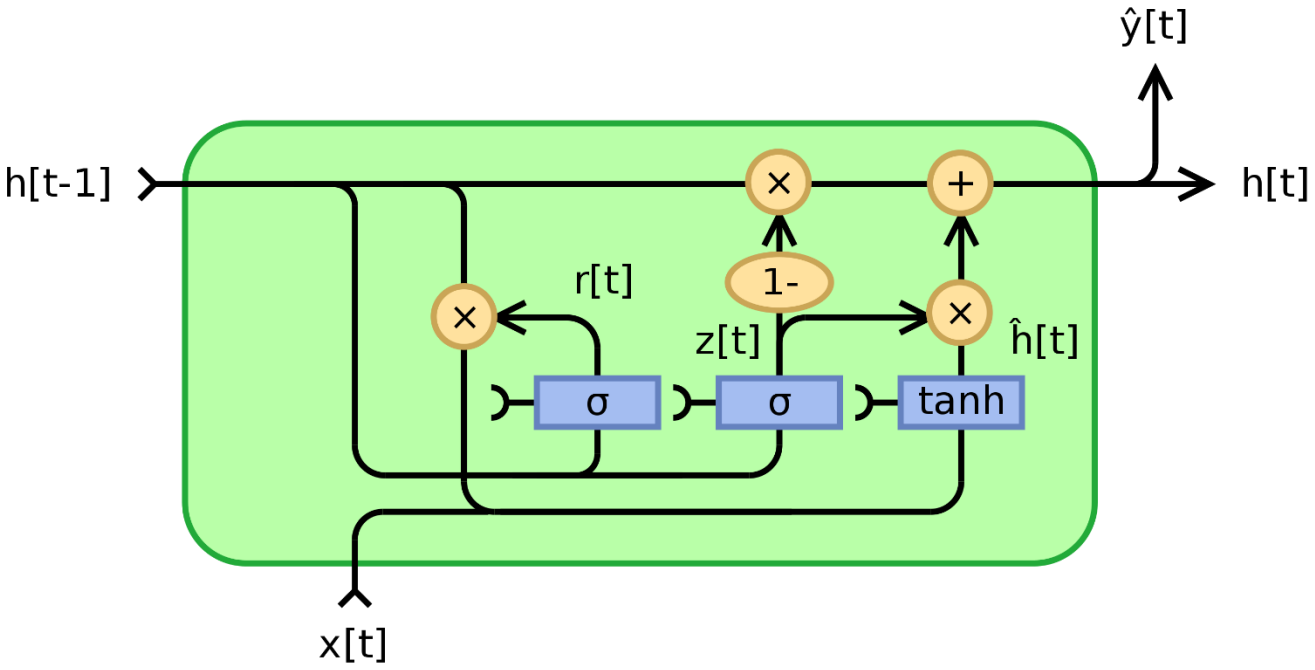


Рисунок 2.5 – схема Fully gated unit Тип 1

$$z_t = \sigma_g(U_z h_{t-1} + b_z) \quad (2.2)$$
$$r_t = \sigma_g(U_r h_{t-1} + b_r)$$

– Тип 2, кожені ворота залежать лише від попереднього прихованого (рис 2.6)

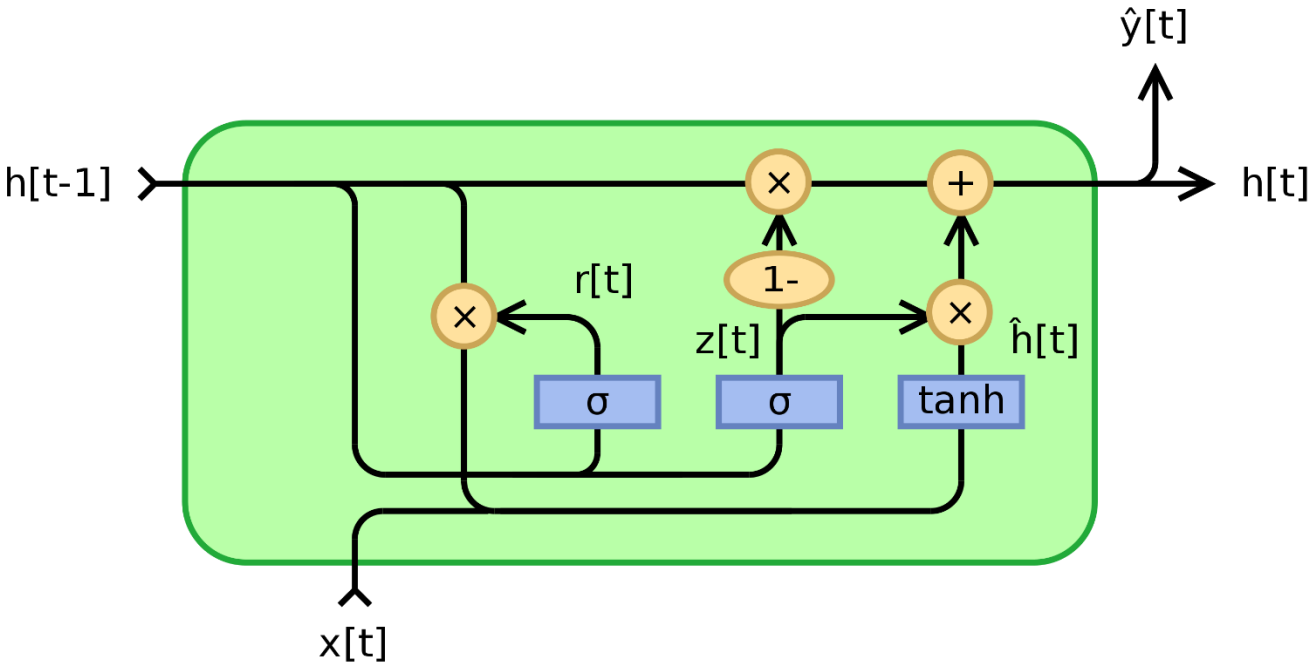


Рисунок 2.6 – схема Fully gated unit Тип 2

$$z_t = \sigma_g(U_z h_{t-1}) \qquad (2.3)$$
$$r_t = \sigma_g(U_r h_{t-1})$$

– Тип 3, кожені ворота залежать лише від зміщення (рис 2.7)

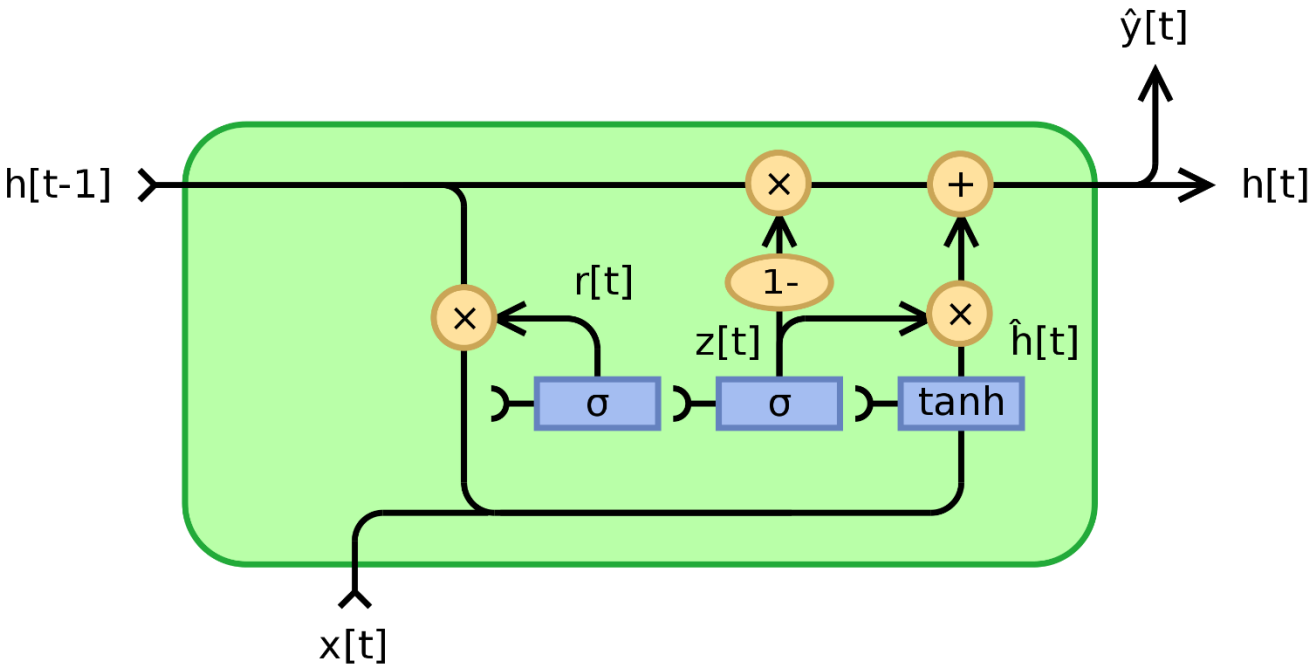


Рисунок 2.7 – схема Fully gated unit Тип 3

$$z_t = \sigma_g(b_z) \\ r_t = \sigma_g(b_r) \tag{2.4}$$

Minimal gated unit схожий з Fully gated unit, за винятком того, що вектор оновлення та скидання об'єднаний в воротах забуття. Це також означає, що рівняння вихідного вектору необхідно змінити.

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\ h_t = f_t \odot h_{t-1} + (1 - f_t) \odot \phi_h(W_h x_t + U_h(f_t \odot h_{t-1}) + b_h) \tag{2.5}$$

Де

- x_t: input vector
- h_t: output vector
- f_t: forget vector
- W, U and b: parameter matrices and vector

2.1.4 Архітектура типу трансформер

До впровадження Трансформаторів більшість найсучасніших систем обробки природних мов спиралися на рекурентні нейронні мережі (RNN), такі як LSTM, з додатковими механізмами уваги. Трансформатор, побудований на цих технологіях уваги, не використовуючи структуру рекурентних нейронних мереж, підкреслюючи той факт, що лише механізми уваги, без повторної послідовної обробки, є достатньо потужними для досягнення продуктивності рекурентних нейронних систем з увагою[8].

Рекурентні нейронні мережі обробляють токени послідовно, підтримуючи вектор внутрішнього стану, який містить данні всіх токенів які

було до теперішнього. Щоб обробити токен номер n, модель поєднує внутрішній стан, що представляє послідовність токенів до токена n-1 з інформацією про новий токен для створення нового прихованого стану, який представляє речення до токену n. Теоретично інформація з одного токену може поширюватися довільно далеко вперед по послідовності, якщо в кожній клітині внутрішній стан продовжує кодувати інформацію про цей токен. На практиці, однак, цей механізм є недосконалим: частково через проблему градієнта, що зникає, прихований стан моделі в кінці довгого речення часто не містить точної, інформації про ранні токени.

Ця проблема була вирішена шляхом запровадження механізмів уваги. Механізми уваги дозволяють моделі безпосередньо дивитись на прихований стан та на будь-який токен до теперішнього. Механізм уваги може отримати доступ до всіх попередніх прихованих станів і зважувати їх відповідно до деякої вивченої міри відповідності поточному токену, надаючи більш чітку інформацію про віддалені відповідні токени. Наочний приклад корисності уваги - в машинному перекладі. В системі машинного перекладу з англійської на французьку мову, найімовірніше, перше слово результату перекладу на французьку мову значною мірою залежить від початку який на англійській мові. Однак у класичній моделі LSTM кодер-декодер, для отримання першого французького слова моделі надається лише вектор стану останнього англійського слова. Теоретично цей вектор може кодувати інформацію про все англійське речення, даючи моделі всі необхідні знання, але на практиці ця інформація часто недостатньо зберігається. Якщо ми запровадимо механізм уваги, модель замість цього може навчитися використовувати приховані стани ранніх англійських токенів при створенні початку французької речення, надаючи їй набагато більш точне зображення того, що вона перекладає.

Коли механізми уваги додаються до рекурентних нейронних систем, він призводить до значного збільшення продуктивності. Введення Трансформатора

виявило той факт, що механізми уваги є потужними самі по собі, і що послідовна періодична обробка даних не потрібна для досягнення продуктивності на рівні з рекурентними нейронними мережами з механізмом уваги. Трансформатор використовує механізм уваги, не будучи рекурентною нейронною мережею, обробляючи всі токени одночасно і обчислюючи ваги уваги між ними. Той факт, що Трансформатори не покладаються на послідовну обробку і дуже легко піддаються паралелізації, дозволяє трансформаторам більш ефективно навчатись на більшій кількості даних.

Трансформатор складається з двох основних компонентів: набору кодувальних ланцюгів і набору декодерів ланцюгів. Функція кожного кодера полягає в обробці його вхідних векторів, щоб генерувати те, що відомо як кодування, які містять інформацію про частини вхідних данних, що мають відношення один до одного. Він передає свій набір згенерованих кодувань наступному кодеру в якості вхідних данних. Кожен декодер робить навпаки, беручи всі кодування та обробляючи їх, використовуючи закодовану контекстну інформацію для створення вихідної послідовності. Щоб досягти цього, кожен кодер і декодер використовує механізм уваги, який для кожного токену зважує відповідність кожного іншого токену і черпає інформацію з них відповідно під час генерування результату. Кожен декодер також має додатковий механізм уваги, який черпає інформацію з результатів роботи попередніх декодерів, перш ніж декодер черпає інформацію з кодувань. І кодери, і декодери мають кінцеву нейронну мережу для додаткової обробки результату, а також містять залишкові з'єднання та етапи нормалізації.

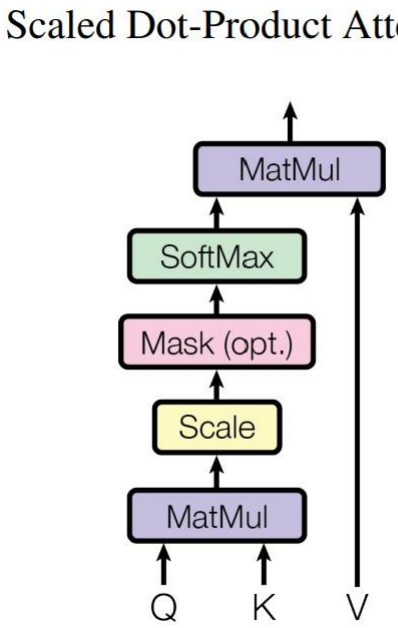


Рисунок 2.8 – механізми уваги

Основні блоки архітектури трансформатора - це механізми уваги (рис. 2.8). Коли речення передається в модель Трансформатора, ваги уваги обчислюються між кожним токеном одночасно. Механізм уваги створює ембедінг для кожного токена в контексті, які містять інформацію не тільки про сам токен, але і зважену комбінацію інших відповідних токенів, зважених вагами механізму уваги.

Конкретно, для кожної одиниці механізму уваги модель Трансформатора вивчає три вагові матриці: ваги W_Q, вагові значення W_K та вагові значення W_V. Для кожного токена i, вхідний ембедінг слова x_i, множимо з кожною з трьох матриць для отримання вектору запиту q_i = x_iW_Q, ключового вектора k_i = x_iW_K та вектора значення v_i = x_iW_V. Ваги уваги обчислюються за допомогою ключових векторів та векторів запиту: вага уваги a_ij a_ij від токена i до токена j - добуток між q_i та k_j. Ваги уваги діляться на квадратний корінь розмірності ключових векторів, sqrt(d_k), який стабілізує градієнти під час тренування, і пропускають через софтмакс, який нормалізує ваги щоб їх сума дорівнювала 1. Той факт, що W_Q і W_K різні матриці дозволяють механізму

уваги бути несиметричним: якщо токен і “звертає увагу” на токен j, це не обов'язково означає, що токен j буде “звертає увагу” на токен і. Результат роботи механізму уваги для токена і - зважена сума векторів значень усіх токенів зважених на a_ij, від і до кожного токена.

Розрахунок уваги для всіх токенів може бути виражений як одне велике матричне обчислення, що корисно для тренувань завдяки оптимізаціям обчислювальної матриці, що робить операції з матрицею швидкими для обчислення. Матриці Q, K і V визначаються як матриці, де і-й рядки є векторами q_i, k_i і v_i відповідно.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^{\text{T}}}{\sqrt{d_k}}\right)V \tag{2.6}$$

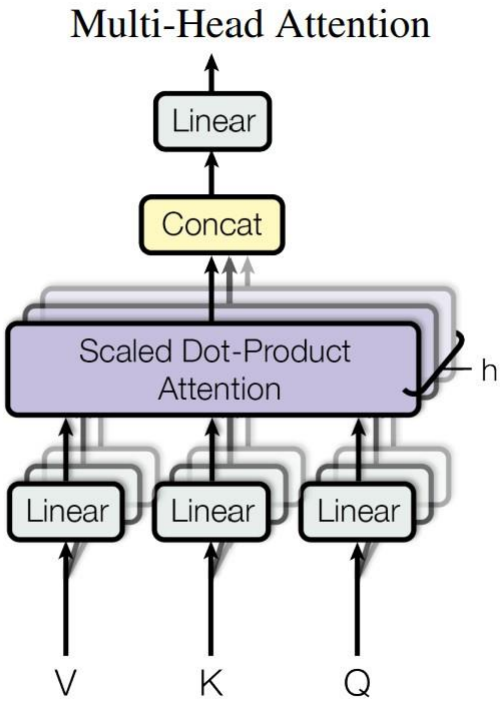


Рисунок 2.9 – Multi-head Attention

Multi-head Attention (рис. 9). Один набір матриць W_Q, W_K, W_V називається головою механізму уваги, і кожен прошарок в моделі

Трансформатора має кілька голів механізму уваги. У той час як одна голова механізму уваги “звертає увагу” на всі токени які "релевантні" до поточного, за допомогою декількох голів механізму уваги модель може навчитися робити це для різних визначень "релевантності". Дослідження показали, що багато голів механізму уваги в Трансформаторах кодують відповідні відносини, прозорі для людини. Наприклад, є голови, що “звертають увагу” на кожний токен, який головним чином є наступним словом, або голови, які в основному “звертають увагу” на дієслова які мають відношення до поточного токену. Оскільки моделі Трансформаторів мають декілька голів механізму уваги, вони мають можливість фіксувати безліч рівнів і типів відповідності, від поверхневого до семантичного. Результат роботи всіх голів в прошарку уваги з'єднуються, щоб переходити в наступний прошарок нейронної мережі.

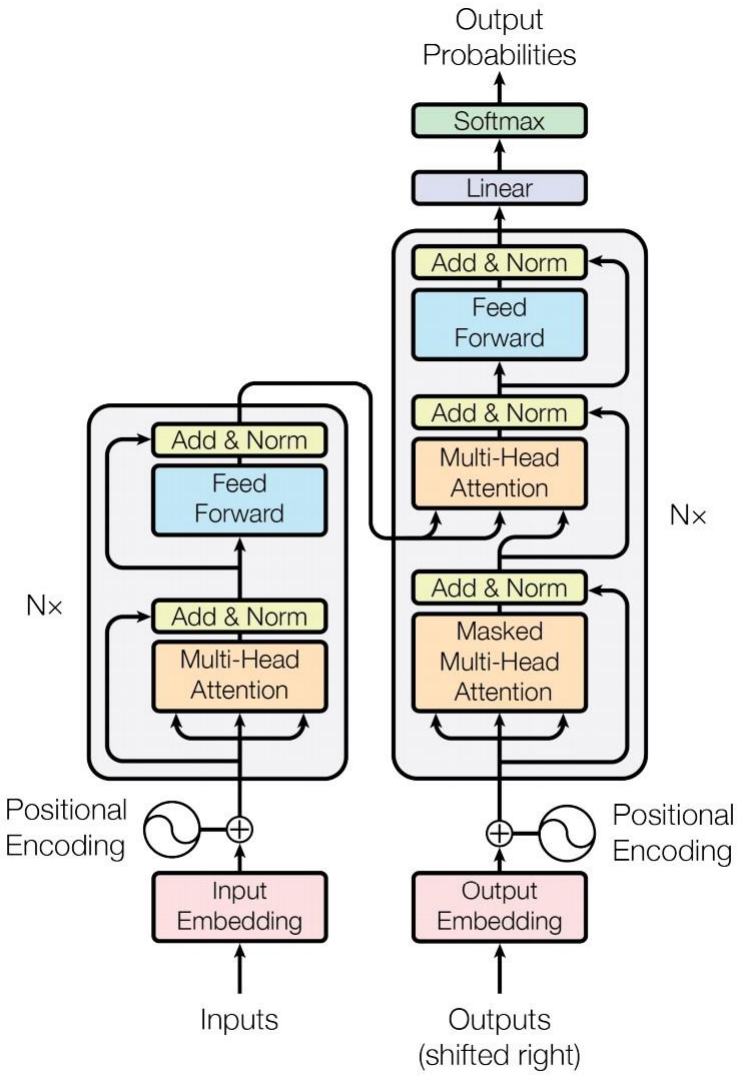


Figure 1: The Transformer - model architecture.

Рисунок 2.10 – Transformer

Encoder (рис. 2.10). Кожен кодер складається з двох основних компонентів: механізму самоуваги та нейронної мережі з прямими зв’язками. Механізм самоуваги приймає набір вхідних кодувань з попереднього кодера і зважує їх відповідність один одному для створення набору кодувань вихідних даних. Потім нейронна мережа з прямими зв’язками, далі обробляє кожне вихідне кодування окремо. Ці вихідні кодування нарешті передаються наступному кодеру як його вхід, а також декодерам.

Перший кодер приймає позиційну інформацію та ембедінги вхідної послідовності як свій вхід, а не кодування. Інформація про положення необхідна, щоб Трансформатор міг використовувати порядок послідовності, оскільки жодна інша частина Трансформатора не використовує цього.

Decoder (рис. 2.10). Кожен декодер складається з трьох основних компонентів: механізму самоуваги, механізму уваги над кодуванням та нейронної мережі з прямими зв’язками. Декодер функціонує аналогічно кодеру, але вводиться додатковий механізм уваги, який черпає інформацію з кодувань, що генеруються кодерами.

Як і перший кодер, перший декодер приймає позиційну інформацію та ембедінги вихідної послідовності як свій вхід, а не кодування. Оскільки трансформатор не повинен використовувати поточний або майбутній вихід, щоб передбачити вихід, послідовність виводу повинна бути частково замаскована, щоб запобігти цьому зворотному потоку інформації. Останній декодер супроводжується остаточним лінійним перетворенням і шаром softmax, щоб створити вихідні ймовірності для кожного слова в словнику.

Трансформатори, як правило, проходять напівавтоматичне навчання з подальшим навчанням з учителем. Преднавчання, як правило проводиться на набагато більшому наборі даних, ніж точна настройка, через обмежену доступність розмічених навчальних даних.

2.1.5 BERT

BERT(Bidirectional Encoder Representations from Transformer)[9] - це новий метод попередньої підготовки представлень мови, який отримує найсучасніші результати для широкого спектру завдань з обробки природних мов (NLP). Типова архітектура BERT складається з 24 послідовних трансформерів(які були описані вище).

2.2 Критерії якості роботи системи / адекватності моделі

2.2.1 Точність роботи на відомих алгоритму сайтах

Перший критерій буде показувати яку з моделей краще обрати для отримання максимальної точності роботи на тих сайтах які були використані для створення моделі. В випадку машинного навчання це – ті сайти на яких модель навчалась, а в випадку фіксованими шляхами до інформації в html структурі сайту – це ті сайти для веб сторінок яких було знайдено шляхи. Тобто модель/алгоритм створюється на основі веб сторінок з певного набору сайтів, а потім результат перевіряється на тих самих сайтах, але на інших веб сторінках. Під точністю будемо вважати число яке дорівнює кількості веб сторінок на яких потрібну інформацію знайдено повністю правильно поділену на кількість всіх веб сторінок

2.2.2 Точність роботи на невідомих алгоритму сайтах

Цей критерій буде показувати яку з моделей краще обрати для отримання максимальної точності роботи на тих сайтах які не були використані для створення моделі. Тобто модель/алгоритм створюється на основі веб сторінок з певного набору сайтів, а потім результат перевіряється на веб сторінках сайтів які не потрапили в цей набір. Під точністю будемо вважати число яке дорівнює кількості веб сторінок на яких потрібну інформацію знайдено повністю правильно поділену на кількість всіх веб сторінок

2.2.3 Необхідність ручного втручання при зміні html структури сайту

Цей критерій буде показувати яку з моделей краще обрати для роботи моделі без ручного втручання з часом. Під необхідністю ручного втручання вважається неспроможність роботи алгоритму/моделі. Зміна html структури сайту може бути спричинена зміною дизайну сайту, зміною або додаванням рекламних банерів на сайті та багатьма іншими факторами які проявляють себе з часом. Цей критерій дискретний і має два значення:

1. Так – тобто при зміні html структури сайту необхідно ручне втручання.
2. Ні – тобто при зміні html структури сайту не необхідно ручне втручання.

2.3 Порівняльний аналіз існуючих методів

2.3.1 Валідаційна стратегія для нейронних мереж

Першим кроком у розробці моделі машинного навчання є навчання та валідація. Для того, щоб підготувати та затвердити модель, ви повинні спочатку розділити ваш набір даних, який передбачає вибір відсотка ваших даних для використання наборів для навчання, валідації та оцінки метрики[13]. Наступний приклад (рис. 2.11) показує набір даних із 64% навчальними даними, 16% валідаційними даними та 20% даними для оцінки метрики.

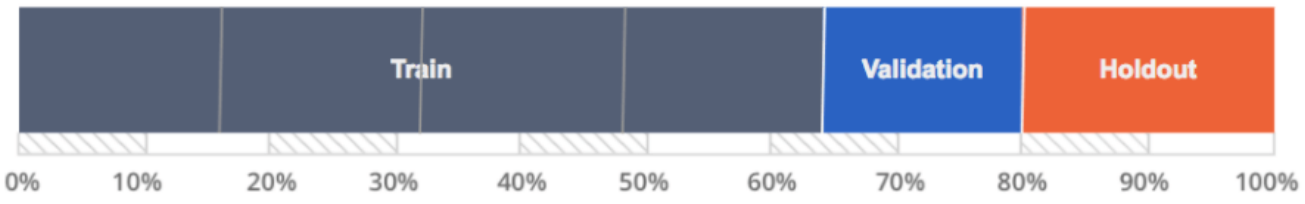


Рисунок 2.11 – валідація

Навчальний набір - це підрозділ набору даних, з якого алгоритм машинного навчання розкриває або «вчиться» зв’язки між функціями та цільовою змінною. У контрольованому машинному навчанні дані про навчання позначаються відомими результатами.

Набір перевірки - це ще один підмножина вхідних даних, до яких ми застосовуємо алгоритм машинного навчання, щоб побачити, наскільки точно він визначає зв’язки між відомими результатами для цільової змінної та іншими функціями набору даних.

Іноді називається даними "тестування", підмножина тримання дає остаточну оцінку ефективності моделі машинного навчання після того, як вона була навчена та затверджена. Набори Holdout ніколи не повинні використовуватися для прийняття рішень про те, які алгоритми використовувати або для вдосконалення або настройки алгоритмів.

Розбиття даних на набори навчань, валідації та витримки дозволяє розробити високоточні моделі, що стосуються даних, які ви збираєте в майбутньому, а не лише даних, на яких навчалася модель. Навчаючи ваші дані, перевіряючи їх та тестуючи їх у наборі виплат, ви отримуєте реальне відчуття того, наскільки точними будуть результати моделі, що призводить до кращих рішень та більшої впевненості у точності вашої моделі.

2.3.2 Порівняння фіксованих шляхів в html структурі та текстового аналізу

Порівнювати будемо у задачі знаходження титулу навини на веб сторінці з новиною. Результати метрики отримані згідно з процесом валідації описаним вище.
Для тестового аналізу буде використано архітектуру нейронних мереж BERT.
Для фіксованих шляхів в html структурі буде використано Xpath.
Результати відображені на таблиці 2.1:

Таблиця 2.1 – Порівняльна таблиця

Назва методу	Точність роботи на відомих алгоритму сайтах	Точність роботи на невідомих алгоритму сайтах	Необхідність ручного втручання при зміні html структури сайту
Фіксованих шляхи в html структурі	1	0	Так
Текстовий аналіз	0.7314	0.5573	Ні

2.4 Алгоритм

За результатами аналізу існуючих рішень проблеми знаходження потрібної інформації на веб сторінці були зроблені наступні висновки:

- Фіксованих шляхи в html структурі не здатні працювати з сайтами для яких не були знайдені шляхи в ручну, проте вони показують ідеальний результат при роботі з сайтами для яких були знайдені ці шляхи. Також потребують постійного ручного оновлення для роботи.
- Текстовий аналіз може працювати з сайтами які були використані при тренуванні моделей та з сайтами які не використовувались для тренуванні моделі, проте демонструють не дуже високу точність. Не потребують ручного втручання з асом.

Так як єдина проблема текстового аналізу це – точність, було прийнято рішення за основу моделі взяти текстовий аналіз. Фіксованих шляхи в html структурі показують гарні результати, але їм катастрофічно не вистачає гнучкості,

тому було прийнято рішення додати до текстового аналізу гнучкий аналіз html структури сторінки.

Архітектура моделі буде виглядати так:

1. В html веб сторінки знаходяться всі блоки текст яких бачить користувач
2. Текст кожного такого блоку аналізується за допомогою методів текстового аналізу (BERT)
3. Html шлях до цього блоку аналізується створенним алгоритмом аналізу html шляхів (ембедінг для кожного html тегу + GRU).
4. Результати роботи(вектори) текстового та html аналізу поєднуються в один вектор та утворюють послідовність з векторів кожен з яких відповідає своєму html блоку
5. Ця послідовність обробляється рекурентною мережею(GRU) результат роботи якої є відранжовані html блоки
6. Вибирається текст html блоку з найбільшим коефіцієнтом

2.5 Висновки

В цьому розділі було розглянуто багато способів роботи з текстовими данними. Кожен має свої переваги та недоліки. Також описано формалізацію абстрактної задачі та загальний підхід до її вирішення через вирішення задачі розмітки послідовності. Для експериментальних досліджень було описано критерії та стратегію оцінки якості результатів роботи моделей.

РОЗДІЛ 3 АНАЛІЗ ПРОГРАМНОГО ПРОДУКТ

3.1 Обґрунтування вибору платформи та мови програмування

3.1.1 Вибір мови програмування

Важко згадати лише одну мову програмування для машинного навчання. Різні опитування та дані показують, що найважливішим фактором при виборі мови є тип проекту, над яким потрібно працювати (область застосування). Для реалізації продукту розглядались три основні мови Python[14], Java[15], R[16].

– Python є лідером, 57% науковців та розробників машинного навчання використовують його, а 33% віддають перевагу іншим мовам для розробок. Не тільки python є широко використовуваною мовою, але це головний вибір для більшості його користувачів завдяки випуску TensorFlow і Pytorch та широкому вибору інших бібліотек. Python - найкращий вибір для початківців у цій галузі. Існує багато бібліотек python, таких як Teano, Keras і scikit-learn, які доступні для машинного навчання, глибокого навчання, штучного інтелекту, NLP тощо. Наприклад: Numpy - це бібліотека, яка допомагає вирішувати багато обчислень. Ще одна причина його популярності полягає в тому, що його синтаксиси дуже прості і їх можна легко засвоїти, завдяки чому алгоритми легко реалізуються. Він надає прямий доступ своїм користувачам для прогнозної аналітики. Це бажана мова для розробників, які прагнуть поставити кращі питання та розширити свої можливості існуючих систем машинного навчання.

– Java є другою найбільш частою мовою, яка використовується при роботі з машинним навчанням; 15% експертів використовують її для захисту мережі / кібер-атак та виявлення шахрайства, де python є менш кращим.

Java - дуже проста у користуванні мова, яка забезпечує простий процес налагодження, величезні послуги пакетів, спрощення роботи у великих проектах, графічне представлення даних та кращу взаємодію з користувачем. Найновіша версія Java, яка є java 11, має покращені можливості для машинного навчання, такі як:

Нові рядкові методи - isBlank, lines, repeat, stripLeading, stripTailing, and strip

Нові файлові методи - writeString, readString і isSameFile

Методи розпізнавання шаблонів - якMatchPredicate тощо.

Java вважається захищеною мовою завдяки використанню байт-коду та пісочниць. Не дивно, що новітні та старіші алгоритми машинного навчання написані на Java. Це функціональна мова програмування, яка дозволить майбутнім системам машинного навчання швидкості та точності.

В корпоративному середовищі краще використовувати Java.

– R - graphic-based мова, що використовується для статистичних обчислень, аналізу та візуалізації в машинному навчанні. Для тих, хто хоче вивчити статистичні дані за допомогою графіків, це ідеальна платформа. Він також використовується для різних цілей науковцями даних у facebook, google та багатьох інших великих компаніях.

  Це дуже популярна мова програмування серед статистиків, а також застосовується до завдань машинного навчання, таких як регресія, класифікація та формування дерева рішень.

  R є найбільш кращим у галузі біоінженерії, біоінформатики та біомедичної статистики. Пакети, які використовуються для машинного навчання: RODBC, Gmodels, Class та TM. Він підходить для разових проектів, таких як доповіді, наукові праці, що включають артефакти.

  Python часто порівнюють з R, ви повинні знати, що ці дві абсолютно різні мови, які використовуються для різних цілей.

За результатом порівння різних мов програмування було обрано мову Python, так як вона найбільш популярна та програмний продукт відповідає тим задачам машинного навчання які найчастіше розробляються на мові Python.

3.1.2 Вибір платформи

Для розробки рішень з використанням нейроних мереж на Python в сучасних реаліях використовують одну з двох найбільш популярних бібліотек Pytorch[17] та TensorFlow[18].

– Pytorch. На основі бібліотеки Torch, PyTorch є бібліотекою машинного навчання з відкритим кодом. PyTorch є обов'язковим, що означає, що обчислення запускаються негайно, значить користувачеві не потрібно чекати, щоб написати повний код, перш ніж перевірити, працює він чи ні. Можна ефективно запустити частину коду та перевірити його в режимі реального часу. Бібліотека побудована на Python для забезпечення гнучкості як платформи розвитку глибокого навчання. Нижче наведено функції, які характеризують PyTorch як бібліоткеу глибокого навчання:
  1) Простий у використанні API
  2) Підтримка Python - PyTorch плавно інтегрується з іншими бібліотеками python для машинного навчання. Схожий на numpy, тому якщо ви вже користуєтесь numpy, ви будете почувати себе як вдома.
  3) Динамічні графіки обчислень - PyTorch надає нам основу для побудови обчислювальних графів під час виконання роботи та навіть їх зміни під час виконання замість заздалегідь визначених графіків із певними функціональними можливостями. Ця можливість цінна в тих випадках, коли ми не знаємо потреби в пам'яті для створення нейронної мережі.

Інші ключові сильні рамки машинного навчання включають:

1) TorchScript: забезпечує безперебійний перехід між графовим режимом та режимом ребер режимом та нетерплячим режимом, щоб прискорити шлях до виробництва.
2) Розподілене навчання: Розподілений сервіс. Pytorch дозволяє оптимізувати продуктивність у дослідженні та виробництві та масштабувати розподілене навчання.
3) Інструменти та бібліотеки: яскрава екосистема інструментів та бібліотек розширює PyTorch та підтримує розвиток у задачах комп'ютерному зору, NLP тощо.

– TensorFlow від Google - відома бібліотека з глибоким навчанням з відкритим кодом для потоку даних та диференційованого програмування для різних завдань. Це символічна бібліотека математики та додатки машинного навчання, такі як нейронні мережі, також використовують цю бібліотеку. Дослідження та виробництво - це основні напрямки використання бібліотеки. Перелічені особливості TensorFlow:
    1) Безпечна побудова моделі: Використовуючи інтуїтивні API високого рівня, такі як Keras, бібліотека дозволяє нам будувати та тренувати моделі ML, що забезпечує швидку ітерацію моделі та просту налагодження.
    2) Створення моделей машинного навчання в будь-якому місці: навчає та розгортає моделі у хмарних сервісах, на попередньому етапі, у веб-переглядачі чи на пристрої незалежно від мови, якою користувач користується.
    3) Надійна експериментація для досліджень: гнучка та зрозуміла архітектура для швидшого перенесення нових ідей від концепції до коду, до сучасних моделей та публікації

За результатами ознайомлення з бібліотеками Pytorch та TensorFlow було прийнято рішення вибору Pytorch, через те, що програмний продукт відповідає тим задачам машинного навчання які найчастіше розробляються на цій бібліотеці.

3.2 Аналіз архітектури програмного продукту

3.2.1 Аналіз архітектури бібліотеки

Програмний продукт представляє з себе бібліотеку для мови програмування Python. За допомогою бібліотеки користувач зможе використати наступні модулі:

- Функція train_model:

  Використовується для навчання моделі на заздалегідь розміченому наборі даних, для отримання моделі яка буде робити передбачення потрібного користувачу тексту на основі датасету який використовувся для навчання. Тобто модель будет намагатися вивчити схему за якою користувач розмітив в данні в тренувальній вибірці.

  Можливо вказати який пристрій використовувати при навчанні модель (наприклад CPU або GPU)

- Клас InfoBlockPredictor

  Містить в собі саму нейронну мережу, функціонал інтернет запитів для отримання html веб сторінок та необхідні алгоритми пре обробки даних для того щоб їх могла використовувати нейронна мережа.

  Має наступні методи:

  - predcit_url

    Метод який використовується для передбачення необхідного тексту. Приймає на вхід адресу веб сторінки на якій треба знайти потрібний текст. В методі відбувається інтернет запит на скачування веб сторінки,

потім отриманий html перетворюється в формат з яким можуть працювати нейронні мережі. Після чого нейронна мережа робить передбачення про відповідність текстів схемі представленій в тренувальному наборі даних для кожного тексту. В кінці вибирається текст з найбільшим показником передбачення від нейронної мережі.

- save

  Використовується для зберігання вагів нейронної мережі

  Приймає на вхід шлях в файловій системі. Результатом роботи є файл з вагами нейронної мережі збережений в файловій системі.

- load

  Використовується для завантаження збережених раніше моделей. Приймає на вхід шлях в файловій системі. Замінює існуючі ваги нейронної мережі на ваги нейронної мережі з файлу.

3.2.2 Аналіз архітектури нейронної мережі

Не будемо освітлювати внутрішній устрій тих блоків нейронної мережі про які було надано інформацію в теоретичній частині.

Текстова покрокова схема роботи нейронної мережі:

- В html веб сторінки знаходяться всі блоки текст яких бачить користувач
- Текст кожного такого блоку аналізується за допомогою методів текстового аналізу (в нашому випадку BERT)
- Html шлях до цього блоку аналізується створеним алгоритмом аналізу html шляхів (ембедінг для кожного html тегу + GRU).
- Результати роботи(вектори) текстового та html аналізу поєднуються в один вектор та утворюють послідовність з векторів кожен з яких відповідає своєму html блоку

– Ця послідовність обробляється рекурентною мережею(GRU) результат роботи якої є коефіцієнт потрібності для кожного html блоку
– Вибирається текст html блоку з найбільшим коефіцієнтом

Схема роботи нейронної мережі (рис 3.1)

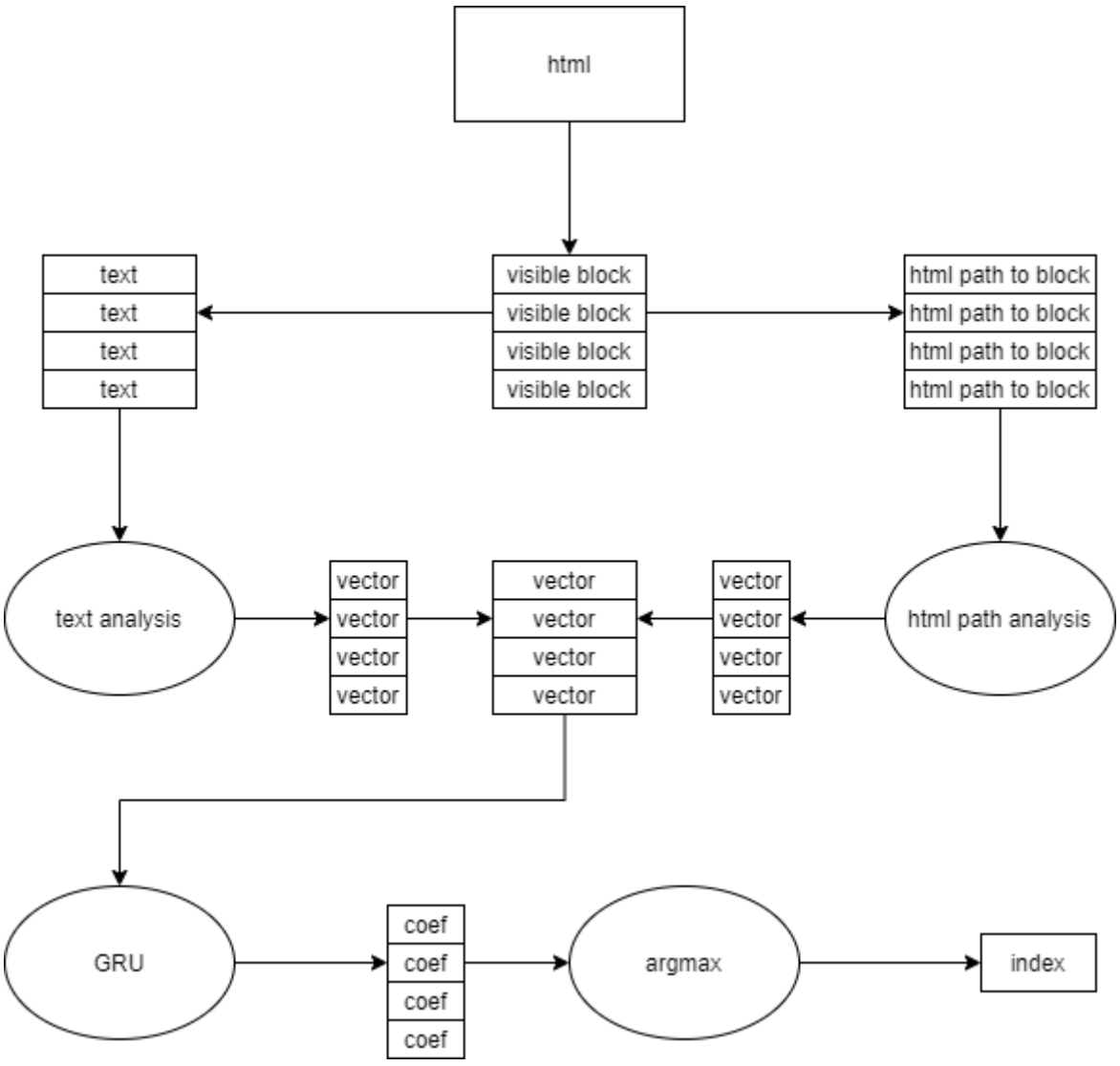


Рисунок 3.1 – Схема роботи нейронної мережі

3.3 Приклади роботи програми

3.3.1 Приклад використання бібліотеки в Jupyter Lab.

Jupyter Lab[19]

```
import pandas as pd
from model_training import train_model
```

```
test_url = 'https://www.bbc.com/news/stories-52731624'
data = pd.read_csv('data/example_data.csv')
data.head()
```

	url	target
0	http://www.latimes.com/business/money/la-fi-mo...	Fed official says weak data caused by weather,...
1	http://www.livemint.com/Politics/H2EvwJSK2VE6O...	Fed's Charles Plosser sees high bar for change...
2	http://www.ifamagazine.com/news/us-open-stocks...	US open: Stocks fall after Fed official hints ...
3	http://www.ifamagazine.com/news/fed-risks-fall...	Fed risks falling 'behind the curve', Charles ...
4	http://www.moneynews.com/Economy/federal-reser...	Fed's Plosser: Nasty Weather Has Curbed Job Gr...

```
model = train_model(data)
```

•••

```
model.predcit_url(test_url)
```

```
'ultraviolet rays could neutralise the virus.'
```

```
model.save('f1', 'f2')
model.load('f1', 'f2')
model.predcit_url(test_url)
```

```
'ultraviolet rays could neutralise the virus.'
```

Рисунок 3.2 – Схема роботи нейронної мережі

На скріншоті (рис 3.2) можна побачити імпортування функцій тренування моделі, підгруздку датасету для тренування, виклик функції тренування моделі,

виклик методу моделі що відповідає за передбачення поьрібної інформації, збереження і завантаження вагів нейронної мережі.

3.3.2 Результати роботи для задачі пошуку інформаційного блоку

Приклади з вірним результатом (рис. 3.3) (рис 3.4) (рис 3.5)

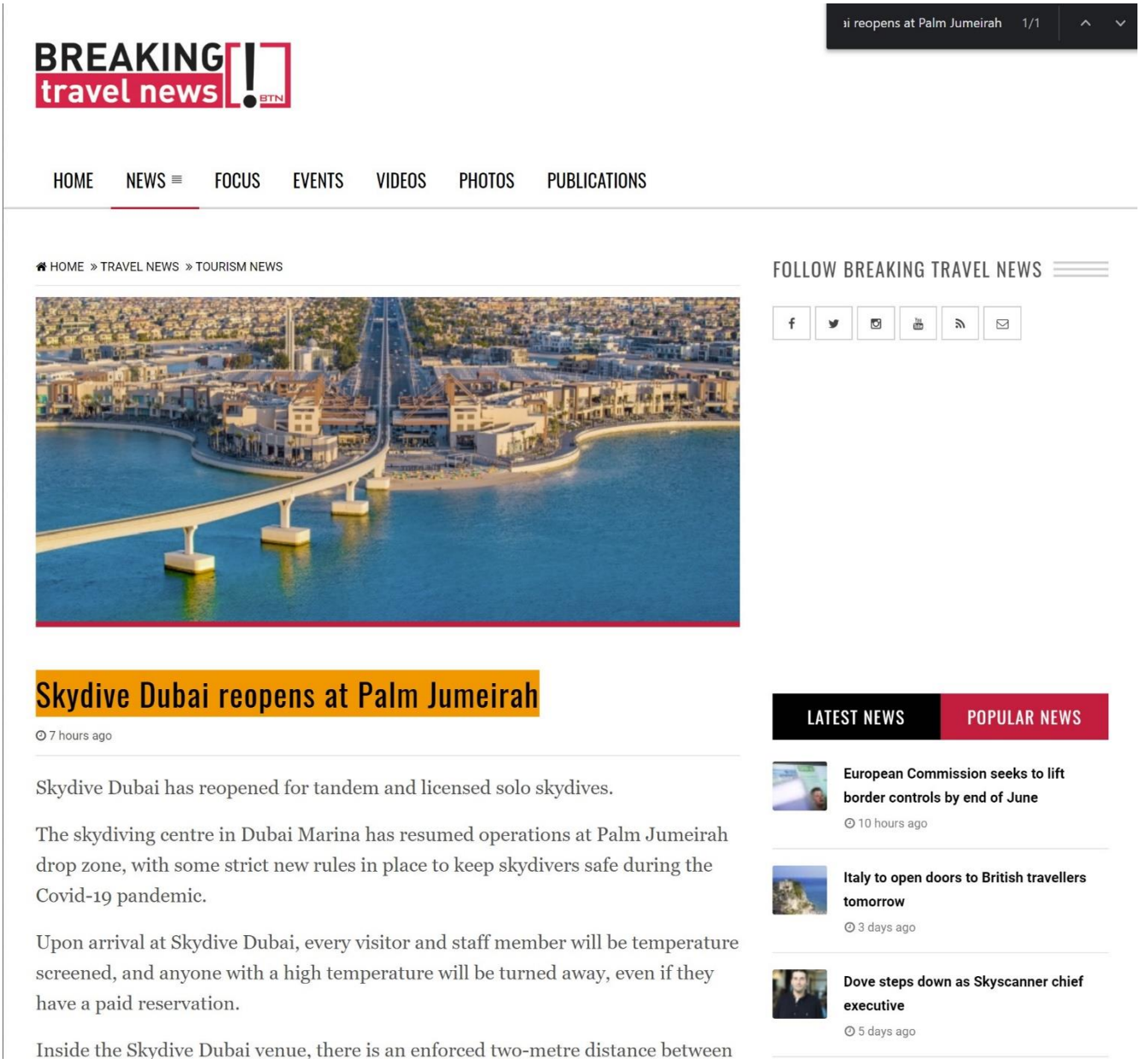


Рисунок 3.3 – Приклади з вірним результатом

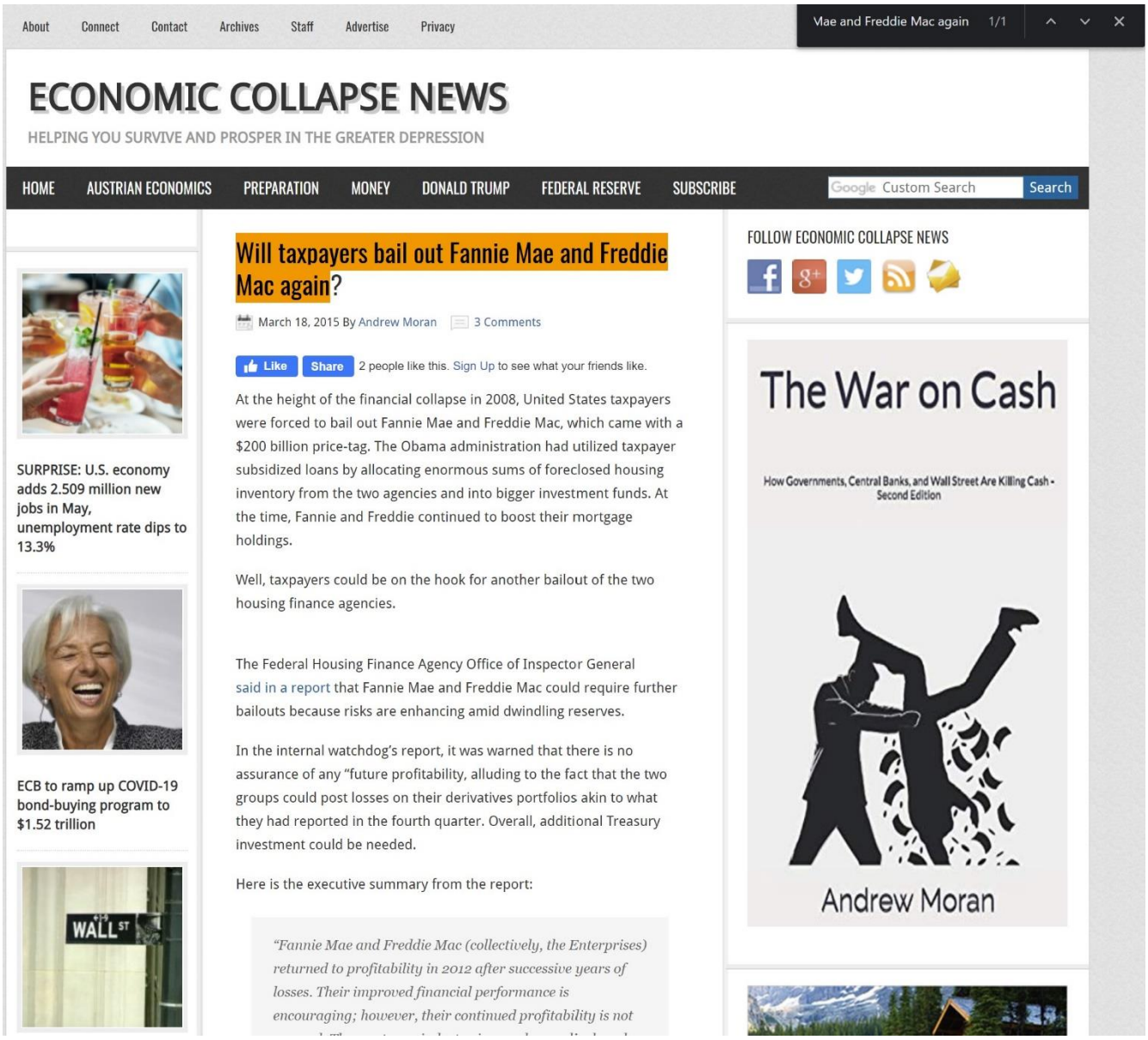
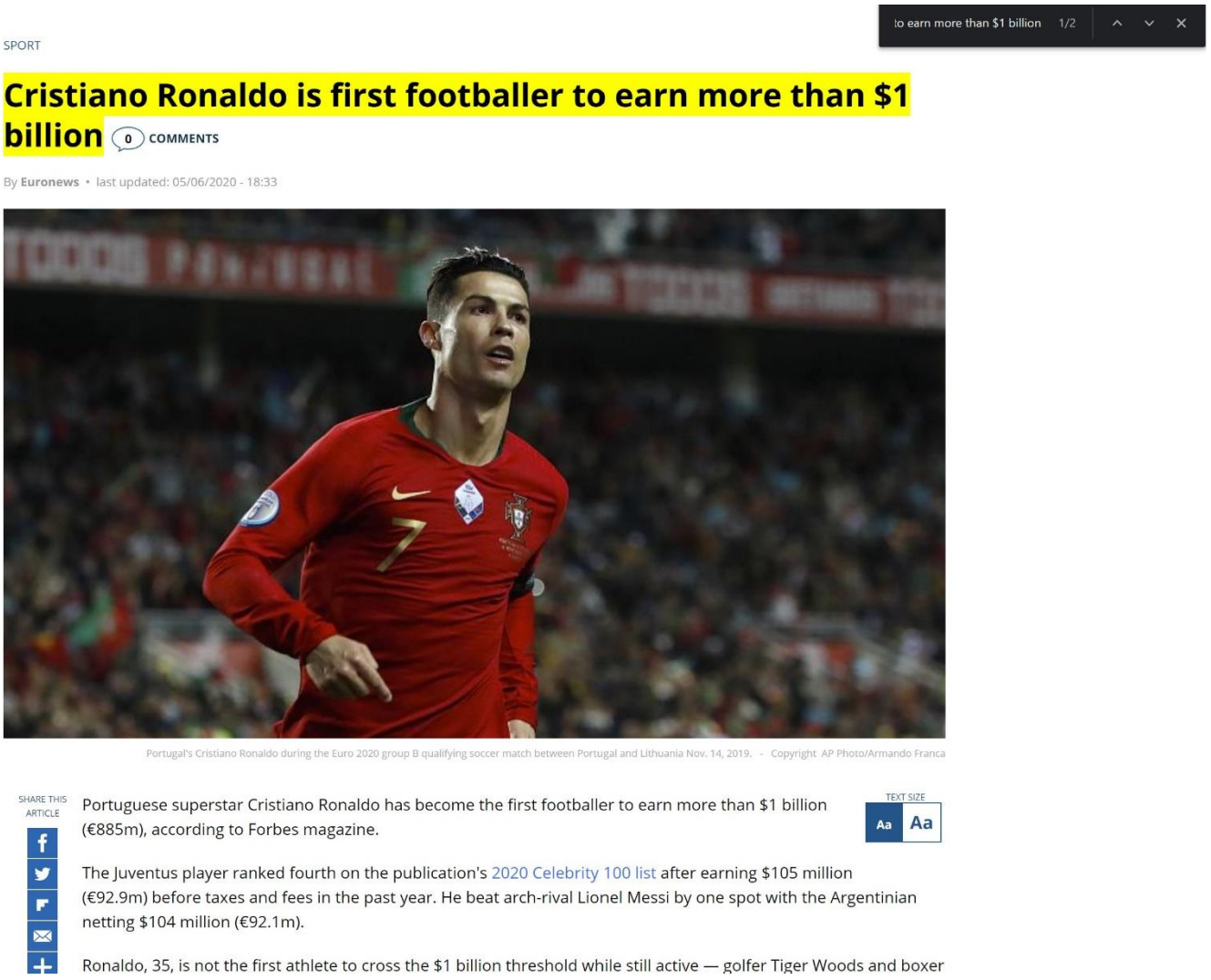


Рисунок 3.4 – Приклади з вірним результатом



to earn more than $1 billion 1/2

SPORT

Cristiano Ronaldo is first footballer to earn more than $1 billion

0 COMMENTS

By Euronews • last updated: 05/06/2020 - 18:33

Portugal's Cristiano Ronaldo during the Euro 2020 group B qualifying soccer match between Portugal and Lithuania Nov. 14, 2019. - Copyright AP Photo/Armando Franca

SHARE THIS ARTICLE

TEXT SIZE Aa Aa

Portuguese superstar Cristiano Ronaldo has become the first footballer to earn more than $1 billion (€885m), according to Forbes magazine.

The Juventus player ranked fourth on the publication's 2020 Celebrity 100 list after earning $105 million (€92.9m) before taxes and fees in the past year. He beat arch-rival Lionel Messi by one spot with the Argentinian netting $104 million (€92.1m).

Ronaldo, 35, is not the first athlete to cross the $1 billion threshold while still active — golfer Tiger Woods and boxer

Рисунок 3.5 – Приклади з вірним результатом

Приклади з не вірним результатом (рис. 3.6) (рис. 3.7)

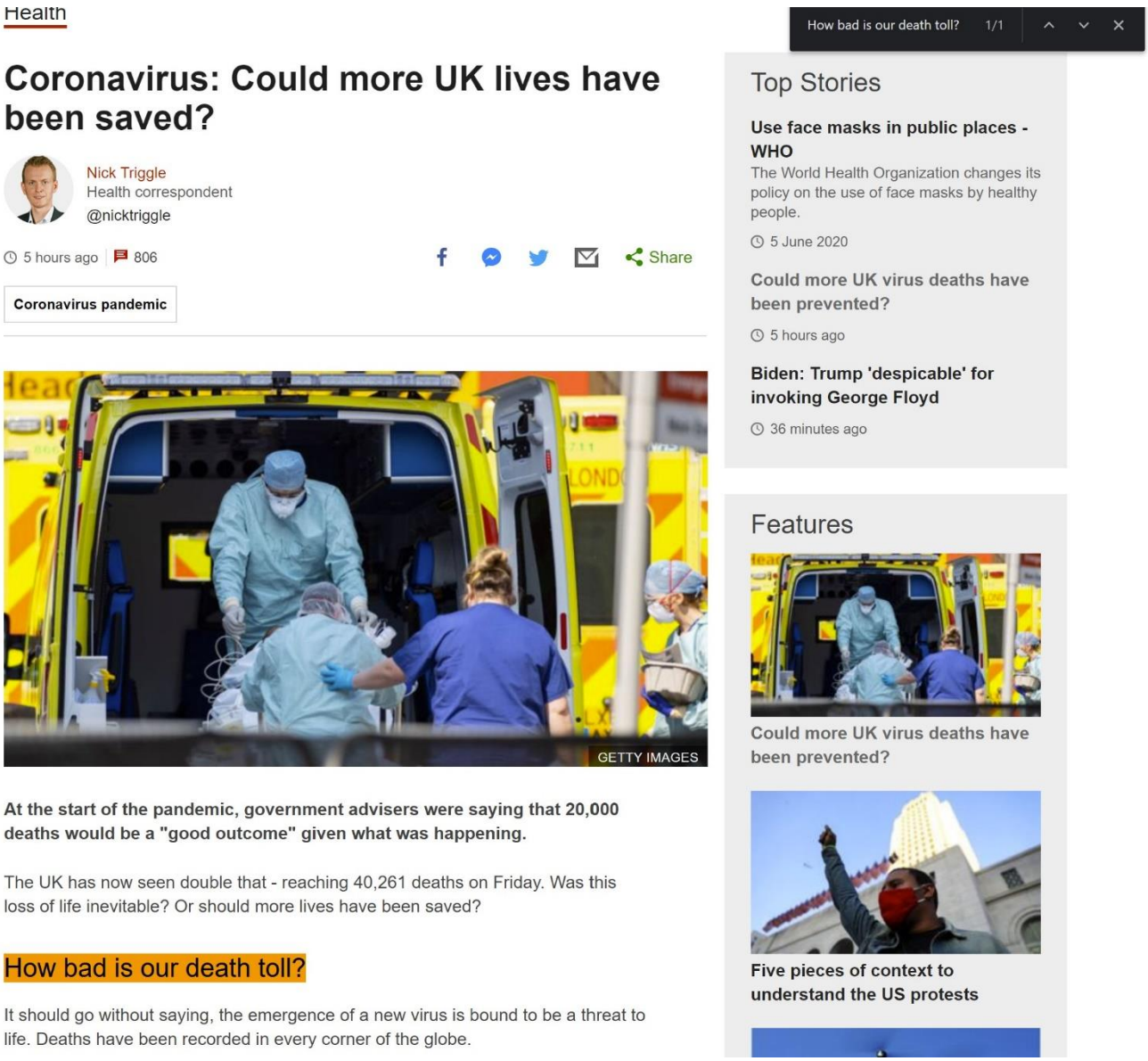


Рисунок 3.6 – Приклади з не вірним результатом

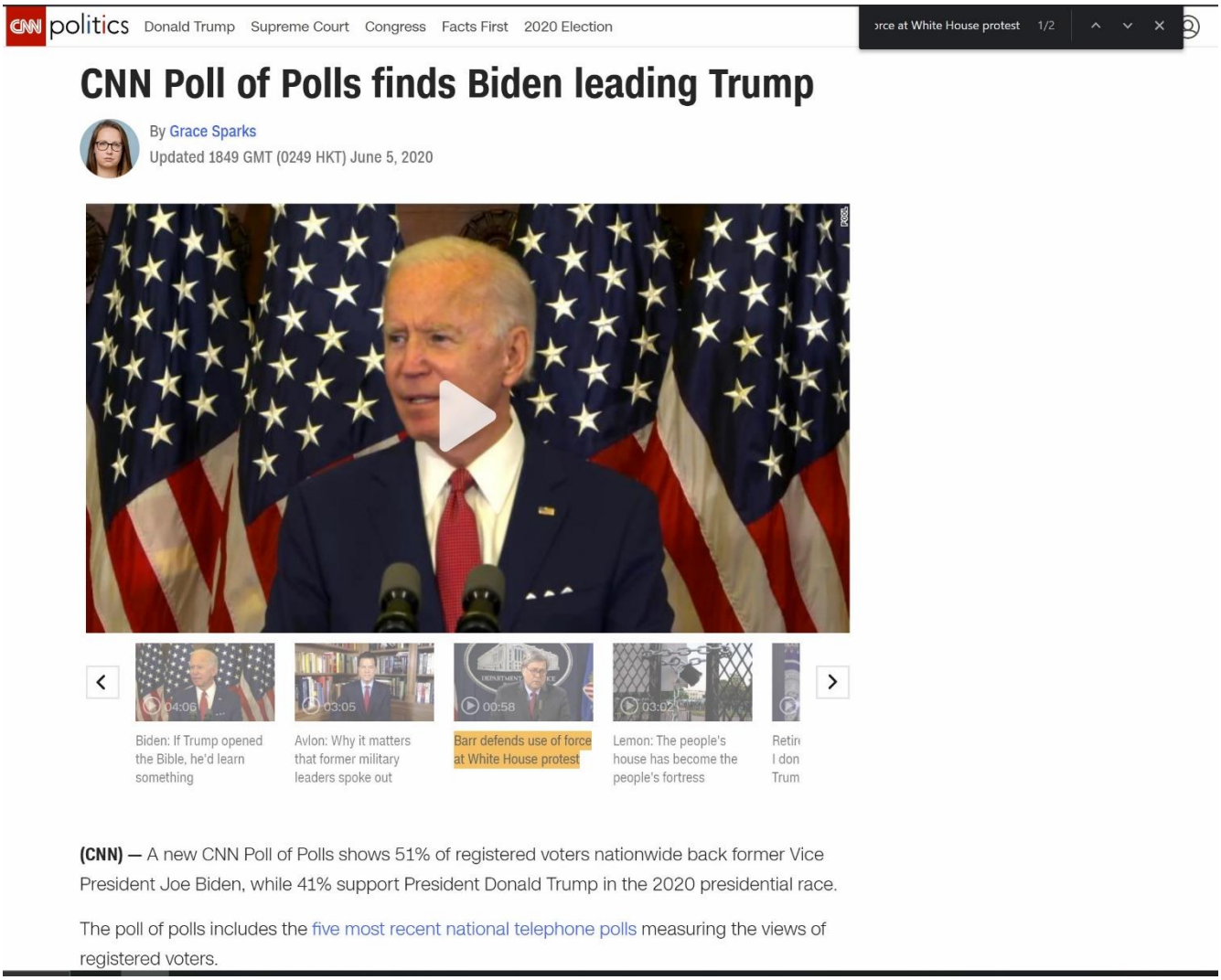


Рисунок 3.7 – Приклади з не вірним результатом

3.4 Аналіз якості роботи системи

Приведемо порівняльну таблицю якості роботи розробленого алгоритму по відношенню до вже існуючих методів за критеріями визначеними при формалізації задачі. Результати отримані на задачі пошуку заголовків новин відображені у таблиці 3.1.

Таблиця 3.1 – Порівняльна таблиця

Назва методу	Точність роботи на відомих алгоритму сайтах	Точність роботи на невідомих алгоритму сайтах	Необхідність ручного втручання при зміні html структури сайту
Фіксованих шляхи в html структурі	1	0	Так
Текстовий аналіз	0.7314	0.5573	Ні
Текстовий аналіз та аналіз структури html	0.9154	0.7342	Ні

3.5 Висновки за розділом

Було розробленно і створенно з використанням програмної мови Python та бібліотеки Pytorch програмний продукт який дозволяє тренерувати та використовувати моделі для пошуку необхідної інформації на веб сторінці.
Програмний продук був використаний для навчання моделі для задачі пошуку заголовків новин на веб сторінках новин та проведений порівняльний аналіз для цієї

задачі з іншими методами. За результатами порівняльного аналізу можна зробити висновки що програмний продукт демонструє кращі результати ніж інші методи.

РОЗДІЛ 4. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ У ЗАДАЧІ ЗНАХОДЖЕННЯ ІНФОРМАЦІЙНОГО БЛОКУ

4.1 Постановка задачі

Проводиться оцінка основних характеристик результатів та їх собівартість. Для отримання результатів використовувалась мова Python. Середовище розробки - VS Code та Jupyter Lab. Робота програми не залежить від технологій реалізації апаратного забезпечення та операційної системи. Нижче приведено аналіз різних підходів до навчання нейронної мережі для задачі знаходження інформаційного блоку на веб сторінці.

4.2 Обґрунтування функцій дослідження

Основні функції:

F_1- вид навчальної вибірки; F_2 - вибір архітектури мережі; F_3 - вибір обчислювального пристрою

Функція F_1: а) датасет “All the news”; б) датасет “News Aggregator Dataset” в) датасет “Bangla Newspaper Dataset”

Функція F_2: а) Аналіз тексту та структури html б) Аналіз тексту

Функція F_3:а) Центральний обчислювальний пристрій (CPU); б) Графічний обчислювальний пристрій (GPU); в) Тензорний обчислювальний пристрій (TPU);

Знизу зображено відповідну морфологічну карту системи (рис 4.1):

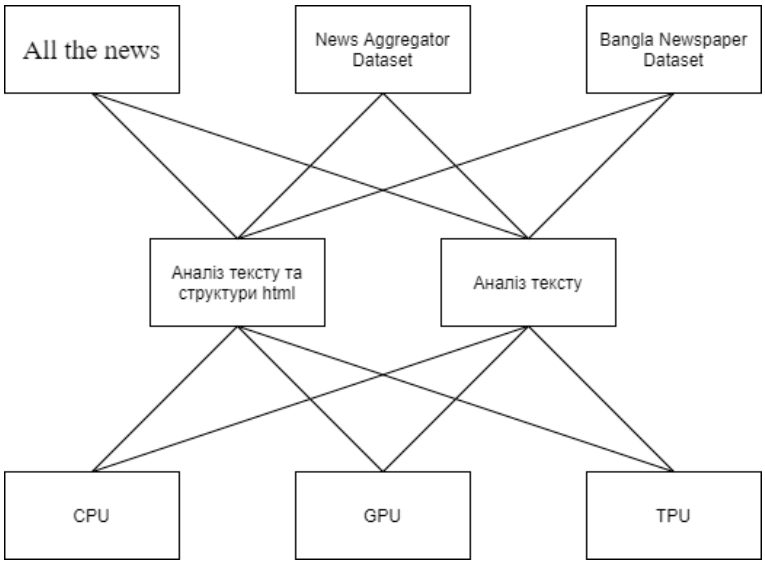


Рисунок 4.1 – Приклади з не вірним результатом

Позитивно-негативна матриця варіантів основних функцій (табл. 4.1):

Таблиця 4.1 – Таблиця позитивно-негативних варіантів

Основні функції	Варіанти реалізацій	Переваги	Недоліки
F_1	А	Містить близько 150 000 новин з детальною інформацією, такою як: заголовок новини, текст новини, автор новини, дата публікації новини, посилання на новину	Новини зібрані з малого переліку сайтів новин(близько 20), що робить дуже складним перевірку якості моделі на сайтах яких вона ще не бачила
F_1	Б	Містить близько 400 000 новин з детальною інформацією,	Відсутній текст новин. В датасеті представлені

		такою як: заголовок новини, дата публікації новини, посилання на новину. Новини зібрані з близько 5000 різних сайтів новин.	новини за 2014 рік.
F_1	В	Містить близько 400 000 новин з детальною інформацією, такою як: заголовок новини, текст новини, автор новини, дата публікації новини, посилання на новину. Новини зібрані з багатьох різних сайтів новин.	Новини на мові Bangla
F_2	А	Існують готові рішення для частини аналітики тесту. Показує більшу точність ніж варіант Б	Потребує розробки власного алгоритму для аналізу структури html
F_2	Б	Існують готові рішення та претренеровані моделі. Потребує меншої обчислювальної сили порівняно з варіантом А.	Потребує розбки власного алгоритму для аналізу структури html.

F_3	А	Доступний в усіх ЕОМ. Достатньо дешевий	Недостатньо потужний для алгоритмів глибокого навчання
F_3	Б	Достатньо потужний для використання для алгоритмів глибокого навчання. Існує широкий інструментарій використання GPU в задачах глибокого навчання	Графічні обчислювальні пристрої – достатньо дорогі та витрачають велику кількість енергії.
F_3	В	Найпотужніший на сьогодні обчислювальний пристрій	Занадто дорогий. Має дуже малий функціонал для роботи з задачами глибокого навчання

Тепер, за наявності позитивно-негативної матриці можна робити висновки щодо доцільності використання одних варіантів та не доцільності використання інших:

На основі порівняльного аналізу варіанті в реалізації основних функцій по їх перевагам та недолікам можна виключити варіанти F_1 А, F_1В, F_3 А, F_3В, тоді варіанти, які залишилися:

F_1 б) - F_2а) - F_3 б)

F_1 б) - F_2 б) - F_3 б)

Для оцінювання описаних функцій запропонована система параметрів. Опишемо цю систему.

Для характеризації досліджень пропонуємо такі параметри:

X_1 — оцінка результату роботи, при невеликому розмірі моделі (середня оцінка (від 1 до 10) поставлена експертами по кожному посиланню на новину)

X_2 - навчання нейронної мережі (час, навчання нейронної мережі);

X_3 - потреби пам'яті обчислювального пристрою(необхідна кількість гігабайт пам'яті);

X_4 — Час на освоєння мови програмування (Час на вивчення мови програмування);

X_5 - складність освоєння бібліотеки pytorch(Час на освоєння бібліотеки pytorch);

X_6 — витрачена електроенергія (кількість кіловат/год для оптимізації алгоритму);

Визначимо, як співвідносяться функції та параметри :

F_1застосовує параметриX_1, X_2, X_3

F_2застосовує параметриX_1, X_2, X_3, X_4, X_5

F_3застосовує параметриX_6

Гірші, середні та кращі показники параметрів вибираються на основі вимог замовника та умов перебігу дослідження, їх наведено у таблиці 4.2:

Таблиця 4.2 – Таблиця умов замовника

Умовні позначення	Одиниці виміру	Гірші	Середні	Кращі
X_1	оцінка	1	5	10
X_2	Год.	144	72	12

X_3	гигабайти	128	24	6
X_4	Год.	400	200	72
X_5	Год.	200	100	36
X_6	кВт/год	100	10	2

Графічні характеристики описаних вище параметрів (рисунок 3.2-3.7):

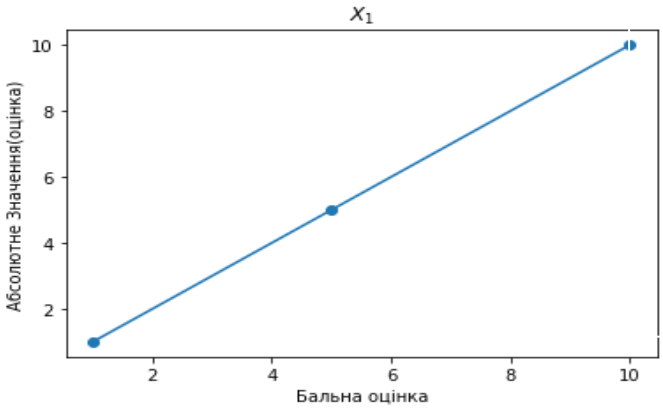


Рисунок 3.2 – X_1

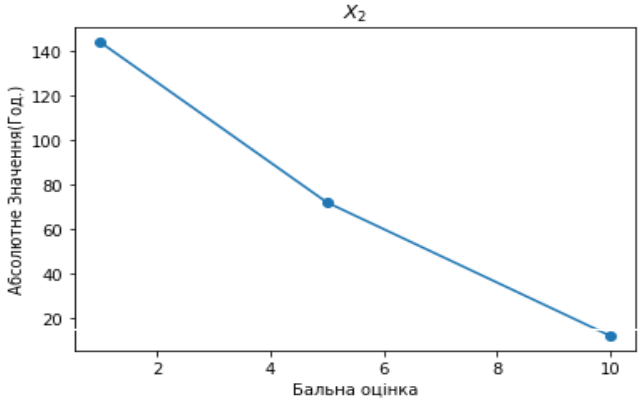


Рисунок 3.3 – X_2

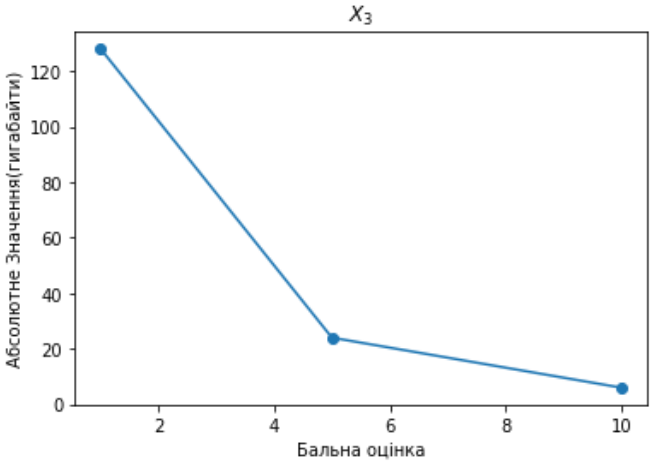


Рисунок 3.4 – X_3

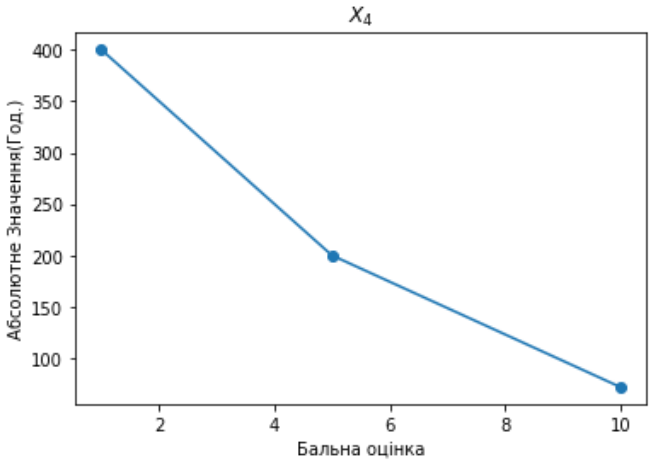


Рисунок 3.5 – X_4

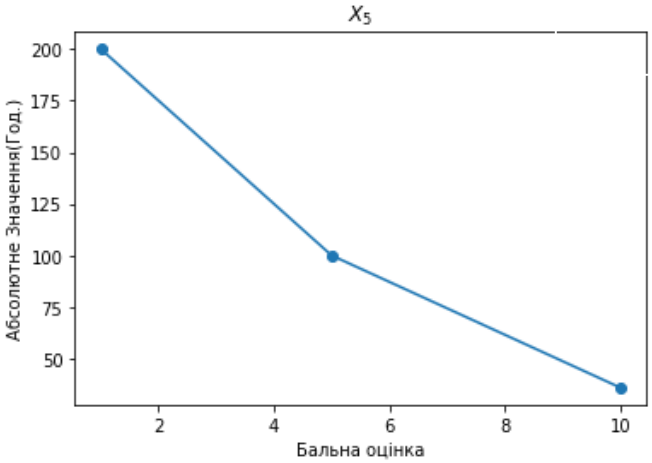


Рисунок 3.6 – X_5

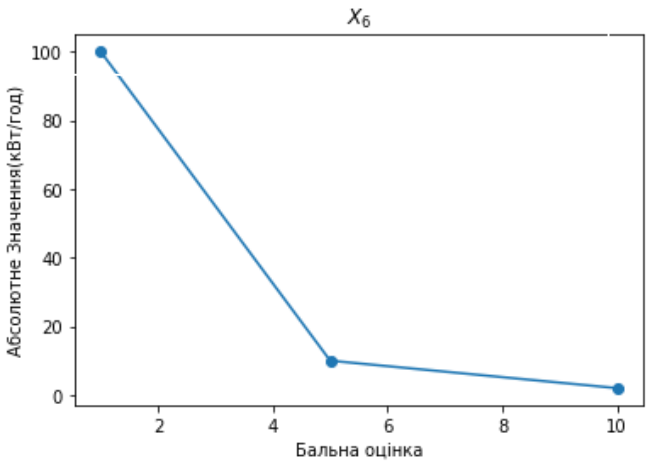


Рисунок 3.7 –X_6

Вагомість параметрів визначається методом попарного їх порівняння на основі результатів ранжування експертами та попарного порівняння параметрів. Наведемо результати експертного ранжування(таблиця 4.3): — достовірне, виходячи із даної нерівності.

Таблиця 4.3 – Таблиця експертного ранжування

Умовнепозначення	Одиницівимірю-вання	Ранг експерта 1	2	3	4	5	6	7	Сумаранів, R_i	Відхи-лення, Δ_i	$\Delta^2{}_i$
X_1	оцінка	1	2	1	1	1	3	1	10	-14.5	210.25
X_2	Год.	3	1	2	3	3	1	2	15	-9.5	90.25

X_3	Год.	2	3	3	2	2	2	3	17	-7.5	56.25
X_4	Год.	6	6	5	6	6	6	4	39	14.5	210.25
X_5	Год.	5	4	6	4	5	4	5	33	8.5	72.25
X_6	кВт/год	4	5	4	5	4	5	6	33	8.5	72.25
Разом		21	21	21	21	21	21	21	147	0	711.5

Визначимо коефіцієнт конкордації:

$$W = \frac{12S}{N^2(n^3-n)} = \frac{12*711.5}{7^2(6^3-6)} \approx 0.8297 > 0.67$$ - достовірне, виходячи із даної нерівності

За найбільший ранг приймаємо 6, за найменший 1

Таблиця 4.4 – Таблиця кінцевої оцінки

Пара-метри	1	2	3	4	5	6	7	Кінцева оцінка	Числове значення

X_1 та X_2	<	>	<	<	<	>	<	<	1.5
X_1 та X_3	<	<	<	<	<	>	<	<	1.5
X_1 та X_4	<	<	<	<	<	<	<	<	1.5
X_1 та X_5	<	<	<	<	<	<	<	<	1.5
X_1 та X_6	<	<	<	<	<	<	<	<	1.5
X_2 та X_3	>	<	<	>	>	<	<	<	1.5
X_2 та X_4	<	<	<	<	<	<	<	<	1.5
X_2 та X_5	<	<	<	<	<	<	<	<	1.5
X_2 та X_6	<	<	<	<	<	<	<	<	1.5
X_3 та X_4	<	<	<	<	<	<	<	<	1.5
X_3 та X_5	<	<	<	<	<	<	<	<	1.5
X_3 та X_6	<	<	<	<	<	<	<	<	1.5
X_4 та X_5	>	>	<	>	>	>	<	>	0.5
X_4 та X_6	>	>	>	>	>	>	<	>	0.5
X_5 та X_6	>	<	>	<	>	<	<	=	1

4.3 Аналіз рівня якості варіантів реалізації функцій

Розрахунок вагомості занотуємо у таблицю 4.5, показники рівня якості у таблицю 4.6:

Таблиця 4.5 – Таблиця розрахунок вагомості

X_i/X_j	X_1	X_2	X_3	X_4	X_5	X_6	$b^1{}_i$	$K^1{}_i$	$b^2{}_i$	$K^2{}_i$	$b^3{}_i$	$K^3{}_i$
X_1	1	1.5	1.5	1.5	1.5	1.5	8.5	0.236	49.75	0.250	273.625	0.251
X_2	0.5	1	1.5	1.5	1.5	1.5	7.5	0.208	41.75	0.210	227.8	0.209
X_3	0.5	0.5	1	1.5	1.5	1.5	6.5	0.180	34.75	0.175	189.62	0.173
X_4	0.5	0.5	0.5	1	0.5	0.5	3.5	0.097	19.75	0.099	109.3	0.100
X_5	0.5	0.5	0.5	1.5	1	1	5	0.138	26.5	0.133	145.75	0.133
X_6	0.5	0.5	0.5	1.5	1	1	5	0.138	26.5	0.133	145.75	0.133
Всього							36	1	199	1	1092	1

Таблиця 4.6 – Таблиця рівня якості

Основні функції	Варіанти реалізацій	Параметри	Абсолютне Значення	Бальна оцінка	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості

				параметра-		
F_1	б)	X_1	7	7	0.251	1.757
		X_2	72	5	0.209	1.045
		X_3	72	3	0.173	0.519
F_2	б)	X_1	6	6	0.251	1.506
		X_2	144	1	0.209	0.209
		X_3	128	1	0.173	0.173
		X_4	200	5	0.100	0.5
		X_5	100	5	0.133	0.665
	а)	X_1	7	7	0.251	1.757
		X_2	72	5	0.209	1.045
		X_3	12	8	0.173	1.384
		X_4	200	5	0.100	0.5
		X_5	100	5	0.133	0.665
F_3	б)	X_6	10	5	0.133	0.665

K_1 = 1.757 + 1.045 + 0.519 + 1.757 + 1.045 + 1.384 + 0.5 + 0.665 + 0.665 = 9.337

K_2 = 1.757 + 1.045 + 0.519 + 1.506 + 0.209 + 0.173 + 0.5 + 0.665 + 0.665 = 7.039

Отже, перший варіант, що передбачає використання Аналізу тексту та структури html дає більший результат. Тому віддаємо йому перевагу.

4.4 Економічний аналіз варіантів розробки ПП

Обидва варіанти включають в себе три окремих етапи:

1 підготовки даних

2 навчання нейронної мережі

- Перший варіант навчання мережі аналізу тексту та мережі аналізу структури html
- Другий варіант навчання тільки мережі аналізу тексту

3 розробка програмного продукту

Для завдання 1 (Алгоритм складності 3, ступінь новизни Г, вид використаної інформації БД)

T_p=12, $K_{\text{п}}$ = 0.3, $K_{\text{СК}}$= 0.7, $K_{\text{СТ.М}}$= 1.2

T_1= 12*0.3*0.7*1.2 = 3.024 людино-днів

Для завдання 2 (при реалізації варіанту 1)), (Алгоритм складності 1, ступінь новизни Б, вид використаної інформації ПІ)

T_p =64, $K_{\text{п}}$ = 2.02, $K_{\text{СК}}$ = 0.8,$K_{\text{СТ.М}}$ = 1.6

$T^1{}_2$ = 64* 2.02 * 0.8 * 1.6 = 165.478 людино-днів

Для завдання 2 (при реалізації варіанту 2)), (Алгоритм складності 1, ступінь новизни Б, вид використаної інформації ПІ)

T_p =64, $K_{\text{п}}$ = 2.02, $K_{\text{СК}}$ = 0.8, $K_{\text{СТ.М}}$ = 1.4

$T^2{}_2$ = 64* 2.02 * 0.8 * 1.4 = 144.793 людино-днів

Для завдання 3(Алгоритм складності 1, ступінь новизни В, вид використаної інформації БД)

T_p =43, $K_{\text{п}}$ = 1.35, $K_{\text{СК}}$ = 0.6,$K_{\text{СТ.М}}$ = 1.5

$T_{\text{З}}$= 43*1.35*0.6*1.5 = 52.24 людино-днів

T_1 = (3.024 + 165.478 + 52.24) * 8 = 1765 людино-годин

T_2= (3.024 + 144.79 + 52.24) * 8 = 1600 людино-годин

В розробці та проведенні дослідження приймають участь 2 програмісти з окладом 12 000 грн

Зарплата розробників становить погодинно:

С = (12000+12000) / (2* 21 * 8) = 71.4 грн

Зарплата поваріантно

C_1 = 1765*71.4 = 126087 грн

C_2 = 1600*71.4 = 114270 грн

Відрахування на соціальний внесок становить 22%:

$C^v{}_1$ = 0.22 * 126087= 27739 грн

$C^v{}_2$ = 0.22 * 114270 = 25139 грн

Визначаємо витрати на оплату однієї машино-години. З урахуванням заробітної плати програміста в розмірі 12000 грн з коефіцієнтом зайнятості 0.2, маємо

$C_{\text{Г}}$ = 12*М*$K_{\text{З}}$ = 12 *12000* 0,2 = 28800 грн

З урахуванням додаткової заробітної плати:

$C_{\text{ЗП}}$ =$C_{\text{Г}}$* (1+ $K_{\text{З}}$) = 28800 * (1 + 0.2)=34560 грн

Відрахування на соціальний внесок:

$C_{\text{ВІД}}$= $C_{\text{ЗП}}$ * 0.22 = 34560 * 0,22 = 7603 грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 8000 грн

$C_{\text{А}}$ = $K_{\text{ТМ}}$* $K_{\text{А}}$*Ц$_{\text{ПР}}$ = 1.15 * 0.25 * 8000 = 2300 грн.

Витрати на ремонт та профілактику можна підрахувати:

$C_{\text{Р}}$ = $K_{\text{ТМ}}$*Ц$_{\text{ПР}}$ * $K_{\text{Р}}$ = 1.15 * 8000 * 0.05 = 460 грн.,

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ} = (Д_К - Д_В - Д_С - Д_Р) * t_З * K_В$ = (365 – 104 – 11 – 16) * 8 * 0.9 = 1684.8

Тепер рахуємо витрати на оплату електроенергії (з урахуванням ПДВ):

$C_{ЕЛ}$ = $Т_{ЕФ} * N_C * К_З * Ц_{ЕН}$ = 1684.8 * 0,3*1,46255 *1. 2 = 884,52 грн.

Накладні витрати рахуються наступним чином:

$С_Н = Ц_{ПР} * 0.67$ = 8000· 0,67 =5360 грн

Річні експлуатаційні витрати:

$С_{ЕКС}$ = 34560 + 7603 + 2300 + 460 + 884.52 + 5360 = 51167.52 грн

Собівартість однієї машино-години ЕОМ становитиме:

$С_{М-Г} = С_{ЕКС} / Т_{ЕФ}$ = 51167.52 /1684.8 = 30.36 грн/год

Витрати на оплату машинного часу складають в залежності від варіанту:

$С^1_М$ = 30.36 * 1765 = 53585.4 грн

$С^2_М$ = 30.36 * 1600 = 48576 грн

Накладні витрати складають 67% від заробітної плати:

$С^1_Н$ = 0.67 * 126087 = 84478 грн

$С^2_Н$ = 0.67 * 114270 = 76560 грн

Таким чином, вартість розробки ПП та проведення дослідів складає:

$С^{❑}_1$ = 126087 + 27739 + 53585.4 + 84478 = 291889.4 грн

$С^{❑}_2$ = 114270 + 25139 + 48576+ 76560 = 264545 грн

Проведемо розрахунок техніко-економічного рівня:

$К_{ТЕР1}$ = 9.337 / 291889.4 = 3.1988 * 10^{-5}

$К_{ТЕР2}$ = 7.039 / 264545 = 2.66 * 10^{-5}

Отже найбільш ефективним є перший варіант з коефіцієнтом техніко економічного рівня 3.1988 * 10^{-5}

За результатами проведеного функціонально-вартісного аналізу, можна зробити висновок, що з альтернатив, що залишилися після відбору, було отримано два варіанти

Таким чином, після першого відбору було отримано два варіанти, в результаті проведення функціонально-вартісного аналізу стало зрозуміло, що доцільним є використання першого варіанту реалізації продукту. першому варіанту відповідає навчання нейронної мережі на датасеті “News Aggregator Dataset ”, використовуючи аналіз тексту та аналіз структури html, використовуючи графічний обчислювальний пристрій (GPU). Така конфігурація є реалізуємою в технічному плані та надає достатньо якісні результати.

ВИСНОВОК

У даній роботі було запропоновано формалізацію задачі пошуку потрібної інформації на веб сторінці і розглянуті основні методи та їх теоретичні основи в цілому та для конкретної задачі.

Було розроблено бібліотеку для мови програмування Python на основі Python бібліотеки Pytorch яка дає можливість використовувати та тренувати моделі з розміченого набору даних для пошуку потрібної інформації на веб сторінці на основі розмітки послідовності html блоків.

Були введені три критерії оцінки - точність роботи на відомих алгоритму сайтах, точність роботи на невідомих алгоритму сайтах, необхідність ручного втручання при зміні html структури сайту. Також була запропонована стратегія валідації цих критеріїв.

Було розглянуто конкретну задачу (пошуку заголовків новин) для якої було реалізовано метод текстового аналізу та метод текстового і html аналізу, проведено порівнювальний аналіз на основі запропонованих критеріїв за результатами якого модель на основі текстового та html аналізу виявилася кращою.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. [Електронний ресурс] – URL: https://html.spec.whatwg.org/multipage/ (дата звернення 14.05.2020)
2. [Електронний ресурс] – URL: https://www.data2type.de/en/xml-xslt-xslfo/xpath/xpath-introduction/ (дата звернення 14.05.2020)
3. [Електронний ресурс] – URL: https://www.w3.org/TR/xslt20/ (дата звернення 14.05.2020)
4. [Електронний ресурс] – URL: http://i.stanford.edu/~ullman/mmds/ch1.pdf (дата звернення 14.05.2020)
5. [Електронний ресурс] – URL: https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html (дата звернення 14.05.2020)
6. [Електронний ресурс] – URL: https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/ (дата звернення 14.05.2020)
7. [Електронний ресурс] – URL: https://www.researchgate.net/publication/13853244_Long_Short-term_Memory (дата звернення 14.05.2020)
8. [Електронний ресурс] – URL: https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf (дата звернення 14.05.2020)
9. [Електронний ресурс] – URL: https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html (дата звернення 14.05.2020)
10. [Електронний ресурс] – URL: https://openai.com/blog/better-language-models/#sample1 (дата звернення 14.05.2020)
11. [Електронний ресурс] – URL: http://papers.nips.cc/paper/5477-neural-word-embedding-as-implicit-matrix-factorization.pdf (дата звернення 14.05.2020)
12. [Електронний ресурс] – URL: https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be (дата звернення 14.05.2020)

13. [Електронний ресурс] – URL: https://www.datavedas.com/holdout-cross-validation/ (дата звернення 14.05.2020)
14. [Електронний ресурс] – URL: https://www.python.org/ (дата звернення 14.05.2020)
15. [Електронний ресурс] – URL: https://www.oracle.com/java/ (дата звернення 14.05.2020)
16. [Електронний ресурс] – URL: https://www.r-project.org/about.html (дата звернення 14.05.2020)
17. [Електронний ресурс] – URL: https://pytorch.org/ (дата звернення 14.05.2020)
18. [Електронний ресурс] – URL: https://www.tensorflow.org/ (дата звернення 14.05.2020)
19. [Електронний ресурс] – URL: https://jupyter.org/ (дата звернення 14.05.2020

ДОДАТОК А ЛІСТІНГ ПРОГРАМНОГО ПРОДУКТУ

fetch_url.py

```
from requests import Session
from requests.adapters import HTTPAdapter
from concurrent.futures import ThreadPoolExecutor, as_completed


REQUEST_TIMEOUT = 3
HEADERS = {'User-Agent': "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.78 "
                         "Safari/537.36", 'Accept-Language': "en, *;q=0.5"}


def fetch_url(url, headers=HEADERS, timeout=REQUEST_TIMEOUT):
    with Session() as s:
        s.mount(url, HTTPAdapter(max_retries=3))
        try:
            resp = s.get(url, headers=headers, timeout=timeout)
            if resp.status_code == 200:
                return {'url': url, 'html': resp.content}
        except:
            pass
        return {'url': url, 'html': None}


def threads_fetch_urls(URLs):
    results = []
    with ThreadPoolExecutor(max_workers=300) as executor:
        futures = [executor.submit(fetch_url, url) for url in URLs]
        for result in as_completed(futures):
            results.append(result.result())
    return results
```

transform_html.py

```
import re
from bs4 import BeautifulSoup, Comment
```

```
def tag_visible(element):
    if element.parent.name in ['style', 'script', 'head', 'title', 'meta', '[document]']:
        return False
    if isinstance(element, Comment):
        return False
    return True

def get_block_text(block):
    text = re.sub(r'\s+', ' ',  str(block))
    text = text.strip()
    if len(text) <= 3:
        return ''
    return text

def get_block_path(block):
    path = []
    block = block.parent
    while block and block.name!='body':
        path.append(re.split(r'\=', block.name)[0])
        block = block.parent
    return path

def block2feature(block):
    text = get_block_text(block)
    path = get_block_path(block)
    return {'text': text, 'path': path}

def transform_html(html):
    if not html:
        return None
    soup = BeautifulSoup(html, 'html.parser', from_encoding='ascii')
    text_blocks = soup.findAll(text=True)
    visible_text_blocks = filter(tag_visible, text_blocks)
    feature = map(block2feature, visible_text_blocks)
    feature = [block for block in feature if block['text']]
    return feature
```

info_block_predictor.py

```
import pandas as pd
import numpy as np

from sentence_transformers import SentenceTransformer

import torch
from torch import FloatTensor, LongTensor
from torch.nn.utils.rnn import pad_sequence
import torch.nn as nn
from torch import cat

from fetch_url import fetch_url
from transform_html import transform_html



class SpatialDropout(nn.Dropout2d):
    def forward(self, x):
        x = x.unsqueeze(2)  # (N, T, 1, K)
        x = x.permute(0, 3, 2, 1)  # (N, K, 1, T)
        x = super(SpatialDropout, self).forward(x)  # (N, K, 1, T), some features are masked
        x = x.permute(0, 3, 2, 1)  # (N, T, 1, K)
        x = x.squeeze(2)  # (N, T, K)
        return x



class Net(nn.Module):
    def __init__(self, emb_size, tag_hidden=128, block_hidden=128):
        super().__init__()
        self.tag_hidden = tag_hidden
        self.block_hidden = block_hidden

        self.dropout_and_act = nn.Sequential(nn.ELU(), SpatialDropout(p=0.2))
        self.block_gru = nn.GRU(batch_first=True, bidirectional=True, input_size=768, hidden_size=block_hidden)
        self.linear = nn.Linear(block_hidden*2, 1)



    def forward(self, texts, paths):
        x = texts.squeeze(dim=0)
        x = x.view(x.shape[0], 1, x.shape[1])
```

```
        x, _ = self.block_gru(x)
        x = self.dropout_and_act(x)

        x = self.linear(x)

        x = x.view(1, -1)
        return x



def tag2id(tag, path2id):
    if tag in path2id.index:
        return path2id[tag]
    return 0
def transform_path2id(feature, path2id):
    return [[tag2id(p, path2id) for p in y['path']] for y in feature]


class InfoBlockPredictor:
    def __init__(self, model=None, path2id=None, text_model=None, device='cuda'):
        self.device = device
        self.model = None
        self.path2id = None
        if (model is not None) and (path2id is not None):
            self.path2id = path2id
            self.model = Net(self.path2id.shape[0]+1).to(device)
            self.model.eval()

        self.text_model = text_model
        if text_model is None:
            self.text_model = SentenceTransformer('distilbert-base-nli-mean-tokens', device=device)


        def load(self, path2id_path, net_path):
        self.path2id = pd.read_csv(path2id_path)
        self.path2id = self.path2id.set_index('Unnamed: 0')['0']

        self.model = torch.load(net_path)
        self.model.to(self.device)
        self.model.eval()
```

```
        return self

    def save(self, path2id_path, net_path):
        self.path2id.to_csv(path2id_path)
        torch.save(self.model, net_path)
        return self


    def predcit_url(self, url):
        fetch = fetch_url(url)
        if fetch['html'] is None:
            print('Internet Error')
            return None
        feature = transform_html(fetch['html'])
        pathes = transform_path2id(feature, self.path2id)
        texts = np.vstack(self.text_model.encode([x['text'] for x in feature]))

        texts_emb = FloatTensor(texts)
        paths = pad_sequence([LongTensor(p) for p in pathes], batch_first=True)
        model_inp = {'texts': texts_emb.to(self.device), 'paths':paths.to(self.device)}
        #return model_inp
        with torch.torch.no_grad():
            block_id = self.model(**model_inp).argmax().item()

        return feature[block_id]['text']
```

model_training.py

```
import pandas as pd
import numpy as np
from itertools import chain

import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pad_sequence
from torch import FloatTensor, LongTensor

from sklearn.model_selection import train_test_split
```

```
import collections

from catalyst.dl import SupervisedRunner
from catalyst.dl.callbacks import AccuracyCallback, AUCCallback, F1ScoreCallback, EarlyStoppingCallback, OptimizerCallback

from sentence_transformers import SentenceTransformer


from info_block_predictor import Net, InfoBlockPredictor
from fetch_url import threads_fetch_urls
from transform_html import transform_html


def temp_filter(line):
    if line['html'] and line['target'] and (line['target'] in str(line['html'])):
        return True
    return False


def get_block_target(line):
    if line['feature']:
        return [int(line['target'] == block['text']) for block in line['feature']]
    return None


def create_path_id_dict(data):
    all_values = data['feature'].map(lambda x:  [y['path'] for y in x])
    all_values = list(chain(*chain(*all_values)))
    all_values = set(all_values)
    all_values = pd.Series(range(1, len(all_values)+1), index=all_values)
    return all_values


def create_text_id_dict(data):
    all_values = data['feature'].map(lambda x:  [y['text'] for y in x])
    all_values = list(chain(*all_values))
    all_values = list(set(all_values))
    idx = range(1, len(all_values)+1)
    all_values = pd.Series(idx, index=all_values)
```

```
    return all_values


def transform_data_for_train(data, text_model):
    #Fetch urls
    fetched = threads_fetch_urls(data['url'])
    fetched = pd.DataFrame(fetched)
    data = data.merge(fetched, on='url')

    #Transform html
    mask = data.apply(temp_filter, axis=1)
    data.loc[~mask, 'html'] = None

    data['feature'] = data['html'].apply(transform_html)
    data = data.drop(columns='html')

    #Finding target
    target = data.apply(get_block_target, axis=1)

    #Filter data
    n_target = target[target.notna()].apply(sum)
    bad_idx = n_target.index[n_target != 1]
    target[bad_idx] = None
    data['block_target'] = target
    data = data[data['block_target'].notna()]
    len_q99 = data['block_target'].map(len).quantile(0.99)
    data = data[data['block_target'].map(len) <=len_q99]

    #Create id for html tags
    path2id = create_path_id_dict(data)
    id_path = data['feature'].apply(lambda x: [[path2id[p] for p in y['path']] for y in x])
    data['path_id'] = id_path

    #Create id for texts'
    text2id = create_text_id_dict(data)
    id_texts = data['feature'].apply(lambda x: [text2id[y['text']] for y in x])
    data['text_id'] = id_texts

    #Create text embeddingss'
    text_embeddings = np.vstack(text_model.encode(text2id.index, batch_size=8*64))
```

```
    text_embeddings = np.vstack([np.array([[0]*768]), text_embeddings])

    data = data.drop(columns=['target', 'feature'])
    return data, path2id, text_embeddings


class InfoBlock_Dataset(Dataset):
    def __init__(self, df, text_embeddings, device='cpu'):
        super().__init__()
        self.texts = df['text_id'].reset_index(drop=True)
        self.paths = df['path_id'].reset_index(drop=True)
        self.target = df['block_target'].reset_index(drop=True)
        self.text_embeddings = text_embeddings
        self.device = device

    def __len__(self):
        return len(self.target)

    def __getitem__(self, idx):
        texts_emb = FloatTensor(self.text_embeddings[self.texts[idx]])

        paths = self.paths[idx]
        paths = pad_sequence([LongTensor(p) for p in paths], batch_first=True)#, enforce_sorted=False)

        target = LongTensor([np.argmax(self.target[idx])]).squeeze(dim=0)
        return {'texts': texts_emb.to(self.device), 'paths':paths.to(self.device), 'targets': target.to(self.device)}



def train_net(data, path2id, text_embeddings, device='cuda', verbose=False):
    train, test = train_test_split(data, test_size=0.2, shuffle=True, random_state=42)
    train, test = train.reset_index(drop=True), test.reset_index(drop=True)
    loaders = collections.OrderedDict()
    loaders["train"] = DataLoader(InfoBlock_Dataset(train, text_embeddings, device=device), shuffle=True)
    loaders["valid"] = DataLoader(InfoBlock_Dataset(test, text_embeddings, device=device), shuffle=True)


    model = Net(path2id.shape[0]+1).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

```
    scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer, milestones=[9], gamma=0.3)

    callbacks = [OptimizerCallback(accumulation_steps=64),
                 AccuracyCallback(),
                 EarlyStoppingCallback(5, metric='accuracy01', minimize=False)]

    runner = SupervisedRunner(device=device, input_key=['texts', 'paths'])

    runner.train(
        model=model,
        criterion=criterion,
        optimizer=optimizer,
        scheduler=scheduler,
        loaders=loaders,
        callbacks=callbacks,
        logdir=None,
        num_epochs=999,
        main_metric="accuracy01",
        minimize_metric=False,
        verbose=verbose,
    )
    return runner.model


def train_model(data, text_model=None, device='cuda'):
    if text_model is None:
        text_model = SentenceTransformer('distilbert-base-nli-mean-tokens', device=device)
    data, path2id, text_embeddings = transform_data_for_train(data, text_model)
    net = train_net(data, path2id, text_embeddings, device)
    ibp = InfoBlockPredictor(net, path2id, text_model, device)
    return ibp
```

ДОДАТОК Б ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДЛЯ ДОПОВІДІ

Знаходження інформаційного блоку на веб сторінці

Автор :
студент 4го курсу групи КА-61
Якубенко Олексій
Керівник:
Доцент, к.т.н. Дідковська М.В.

АКТУАЛЬНІСТЬ РОБОТИ

Зумовлена необхідністю автоматизувати роботу з веб сторінками в таких задачах як

- Знаходження тексту новини на сайтах новин для агрегації новин
- Знаходження опису та ціни продукту на сайтах конкурентів для автоматичного заповнення бази даних продукції та корегування ціни

ПОСТАНОВКА ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

- Проаналізувати існуючі методи пошуку потрібної інформації на веб сайті
- Проаналізувати можливості сучасних методів текстового аналізу у цій задачі
- Розробити алгоритм для гнучкого аналізу html структури веб сторінки
- Розробити програмний продукт для створення інструменту який буде знаходити необхідну користувачу інформацію на сторінці

АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ЗНАХОДЖЕННЯ ІНФОРМАЦІЙНОГО БЛОКУ НА ВЕБ СТОРІНЦІ

- Фіксовані шляхи в html структурі сайту
  - Необхідне ручне втручання з часом
- Аналіз тексту
  - Низька точність

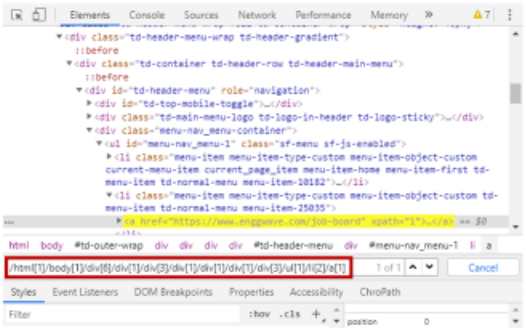
АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ЗНАХОДЖЕННЯ ІНФОРМАЦІЙНОГО БЛОКУ НА ВЕБ СТОРІНЦІ

- Фіксовані шляхи в html структурі сайту
  - Необхідне ручне втручання з часом
- Аналіз тексту
  - Низька точність

Фіксовані шляхи в html структурі сайту

Ідея у представленні html документа у вигляді дерева.

- З часом html структура сайту змінюється
- Кожен сайт зазвичай має свою унікальну html структуру
- Дуже висока швидкість роботи



Текстовий Аналіз

Зазвичай базується на нейронних мережах

- Для навчання потрібно заздалегідь підготовленні данні
- Потребує дуже великої кількості даних для точної роботи
- Неможливо отримати сто відсотків правильні відповіді
- Здатний вирішувати задачі які класичні методи вирішити не можуть

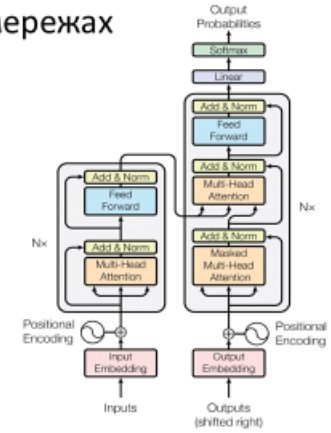


Figure 1: The Transformer - model architecture.

Критерії

$$\text{Точність} = \frac{\text{Кількість веб сторінок для яких правильно знайдений текст в тестовій вибірці}}{\text{Кількість веб сторінок в тестовій вибірці}}$$

- Точність роботи на відомих алгоритму сайтах
- Точність роботи на невідомих алгоритму сайтах
- Необхідність ручного втручання при зміні html структури сайту

Кожен критерій показує яку з моделей краще обрати для роботи з певним класом задач

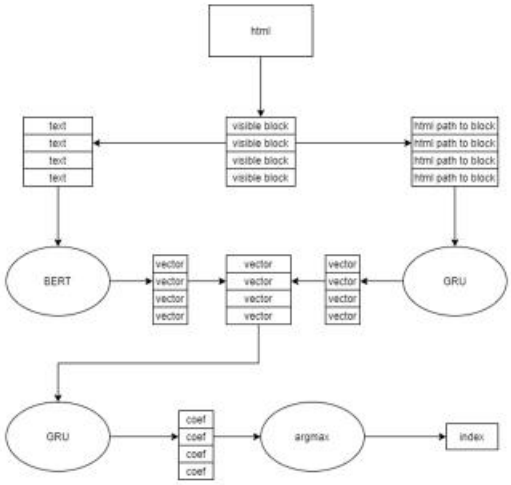
ТОЧНІСТЬ РОБОТИ СИСТЕМИ

Назва методу	на відомих сайтах	на не відомих сайтах	ручне втручання
Фіксовані html шляхи	100%	0%	Так
Аналіз тексту	73%	55%	Ні
Аналіз тексту та структури html	91.5%	73%	Ні

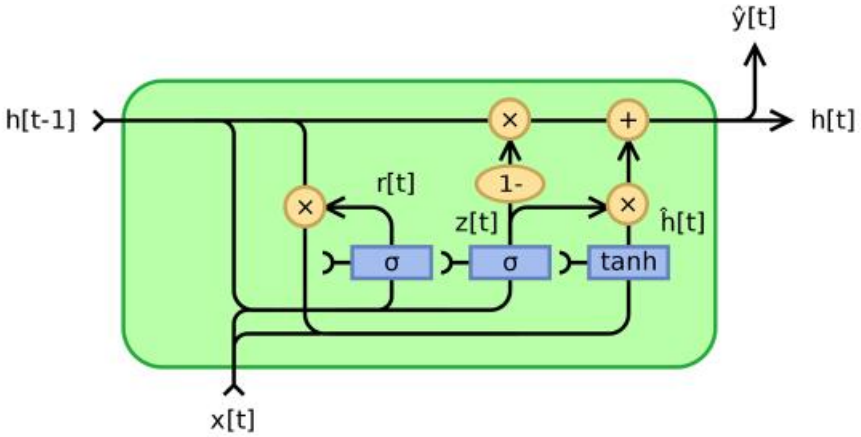
Алгоритм

1) В html веб сторінки знаходяться всі блоки текст яких бачить користувач
2) Текст кожного такого блоку аналізується за допомогою методів текстового аналізу (в нашому випадку BERT)
3) Html шлях до цьго блоку аналізується створеним алгоритмом аналізу html шляхів (в нашому випадку ембедінги + GRU)
4) Результати роботи(вектори) текстового та html аналізу поєднуються в один вектор та утворюють послідовність з векторів кожен з яких відповідає своєму html блоку
5) Ця послідовність обробляється рекурентною мережею (в нашому випадку GRU) результат роботи якої є коефіцієнт потрібності для кожного html блоку
6) Вибирається текст html блоку з найбільшим коефіцієнтом

Архітектура



Архітектура GRU



Архітектура BERT

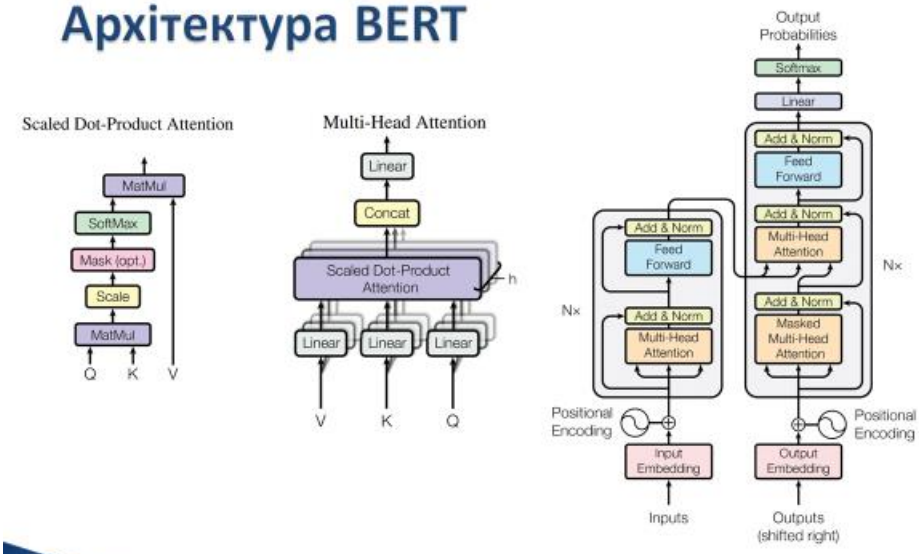


Figure 1: The Transformer - model architecture.

Приклад результату роботи на задачі пошуку заголовку новини

Правильно знайдено заголовок новини

Неправильно знайдено заголовок новини

Приклад використання

```
import pandas as pd
from model_training import train_model
```

```
test_url = 'https://www.bbc.com/news/stories-52731624'
data = pd.read_csv('data/example_data.csv')
data.head()
```

	url	target
0	http://www.latimes.com/business/money/la-fi-mo...	Fed official says weak data caused by weather,...
1	http://www.livemint.com/Politics/H2EvwJSK2VE6O...	Fed's Charles Plosser sees high bar for change...
2	http://www.ifamagazine.com/news/us-open-stocks...	US open: Stocks fall after Fed official hints ...
3	http://www.ifamagazine.com/news/fed-risks-fall...	Fed risks falling 'behind the curve', Charles ...
4	http://www.moneynews.com/Economy/federal-reser...	Fed's Plosser: Nasty Weather Has Curbed Job Gr...

```
model = train_model(data)
```

```
***
```

```
model.predcit_url(test_url)
```

```
'ultraviolet rays could neutralise the virus.'
```

```
model.save('f1', 'f2')
model.load('f1', 'f2')
model.predcit_url(test_url)
```

```
'ultraviolet rays could neutralise the virus.'
```

ВИСНОВКИ

- Запропоновано архітектуру нейронної мережі для пошуку потрібної інформації на веб сторінці
- Розроблено програмний продукт для створення та тренування нейронної мережі пошуку потрібної інформації на веб сторінці для різних типів інформації
- За допомогою програмного продукту реалізовано та натреновано нейронну мережу для пошуку заголовків новин яка показує 91.5% точності

ДЯКУЮ ЗА УВАГУ